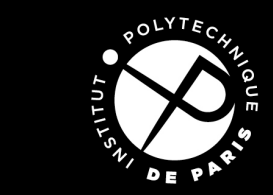
Contributions to Econometric and Deep Learning Methods for Time Series Forecasting

Hugo Inzirillo



INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAG005

Thèse de doctorat

Contributions to Econometric and Deep Learning Methods for Time Series Forecasting

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'Ecole National de la Statistique et de l'Administration Economique

École doctorale n°574 École doctorale de mathématiques Hadamard (EDMH)
Spécialité de doctorat : Mathématiques appliquées

Thèse présentée et soutenue à Palaiseau, le 23 octobre 2024, par

HUGO INZIRILLO

Composition du Jury :

Olivier Lopez Professeur, l'Ecole National de la Statistique et de l'Administration Economique (CREST)	Président
Romuald Elie Professeur, Université Paris-Est (LAMA)	Rapporteur
Frédéric Abergel Chercheur, Fondation pour la Recherche sur la Biodiversité	Rapporteur
Olivier Guéant Professeur, Université Paris 1 Panthéon Sorbonne (CES)	Examinateur
Jean-David Fermanian Professeur, l'Ecole National de la Statistique et de l'Administration Economique (CREST)	Directeur de thèse

Acknowledgements

Humbest thanks to those who have made this thesis possible through their unconditional support. In particular, I would like to express my deep gratitude to my parents, grandparents and wife for their constant presence and unfailing love. Never forget that your encouragement enabled me to overcome the challenges I encountered. Eternal gratitude for your sacrifices and unwavering faith in me. Immensely grateful, I consider this achievement to be the fruit of our joint efforts. For these reasons, I hereby dedicate this thesis to you.

My thanks go first and foremost to Jean-David Fermanian, my thesis supervisor, for his support and benevolence over the last few years. I contacted you by e-mail to discuss a thesis project, and our common interest in digital assets enabled us to come up with an exciting project. I was extremely lucky to have you as my thesis supervisor, and I sincerely thank you for your guidance and for allowing me to excel. Your presence enabled me to overcome many challenges and bring this thesis to a successful conclusion. Thank you so much for your invaluable help, it was an honor

To you, my love, it is with immense gratitude that I thank you for all you have done. In every difficult moment, you have been my light and my guide. Through the ups and downs, your unfailing support has sustained me. The love you've given me is priceless and precious. The moments we shared are unforgettable. No one could replace your place in my life. Eternally grateful, I thank you from the bottom of my heart.

Many thanks to all the members of the CREST Laboratory, for your availability and kindness. Thanks to you, I was able to complete my thesis in optimal conditions.

I would also like to thank Alexandre Coutouly and Adime Amoukou for their support and help during my time with the company. Your support was essential in helping me complete my thesis in optimum conditions.

My final thanks go to Rémi Genet and Benjamin Mat, my classmates, friends and co-authors. Thank you for your unfailing support and for the good times we shared together.

Remerciements

Humbles remerciements à ceux qui ont rendu cette thèse possible par leur soutien inconditionnel. En particulier, je veux exprimer ma profonde gratitude à mes parents, mes grands-parents et ma femme pour leur présence constante et leur amour sans faille. Ne jamais oublier que votre encouragement m'ont permis de surmonter les défis rencontrés. Reconnaissance éternelle pour vos sacrifices et votre foi indéfectible en moi. Immensément reconnaissant, je considère cet accomplissement comme le fruit de nos efforts communs. Pour ces raisons, je vous dédit cette thèse.

Mes remerciements vont tout d'abord à Jean-David Fermanian, mon directeur de thèse, pour son soutien et sa bienveillance tout au long de ces dernières années. Je t'ai contacté par mail pour discuter d'un projet de thèse, et notre intérêt commun pour les actifs numériques nous a permis de faire émerger un projet passionnant. J'ai eu énormément de chance de t'avoir comme directeur de thèse, et je te remercie sincèrement pour ton encadrement et pour m'avoir permis de me surpasser. Ta présence m'a permis de surmonter de nombreux défis et de mener à bien cette thèse. Merci infiniment pour ton aide précieuse, ce fut un honneur.

A toi, mon amour, c'est avec une gratitude immense que je te remercie pour tout ce que tu as fait. Dans chaque moment difficile, tu as été ma lumière et mon guide. À travers les hauts et les bas, ton soutien indéfectible m'a soutenu. L'amour que tu m'as donné est inestimable et précieux. Inoubliables sont les moments que nous avons partagés. Nul ne pourrait remplacer ta place dans ma vie. Éternellement reconnaissant, je te remercie du fond du cœur.

Un grand merci à tous les membres du Laboratoire du CREST, pour votre disponiblité et bienveillance. Grâce à vous, j'ai pu réaliser ma thèse dans des conditions optimales.

Je remercie également Messieurs Alexandre Coutouly et Adime Amoukou pour leur soutien et leur aide lors de ma période en entreprise. Votre appui a été essentiel pour mener à bien ma thèse dans des conditions optimales.

Mes derniers remerciements vont à Messieurs Rémi Genet et Benjamin Mat, mes camarades de promotion, amis et co-auteurs. Je vous remercie pour votre soutien indéfectible et pour les bons moments que nous avons partagés ensemble.

Abstract

This thesis, structured in four distinct parts, contributes to enriching the fields of deep learning and nonlinear econometrics. Traditional models have often shown weaknesses in capturing the non-linear behaviors and regime shifts characteristic of digital assets. This thesis focuses on improving the prediction of returns and volatility of this new asset class through the development of new neural network architectures and innovative methodologies. In a first part, we introduce Temporal Kolmogorov-Arnold Networks (TKANs) as well as a new transformer-based architecture (TKAT) based on TKANs for time series forecasting. These models integrate memory management and attention mechanism to capture the complex dynamics of financial markets. Our results show that TKAN and TKAT outperform traditional recurrent models (LSTM, GRU) in terms of stability and prediction accuracy, particularly over longer time horizons. The second part proposes a new approach for modeling market regimes using recurrent neural networks and new Markov-Switching GARCH dynamics. This former approach allows us to detect market regimes and estimate transition probabilities between different states. The results show a significant improvement in the detection and prediction of market regimes compared to traditional models. The latter models define new meaningful MS-GARCH specifications and convenient simulation-based estimation procedures. In the third part, we explore the use of signatures for portfolio construction. A signature-based classification method is developed to select the most representative assets of each cluster, thus improving the risk-return ratio. Portfolios built with this methodology show superior performance compared to standard portfolios. The last part of this thesis addresses volatility modeling and anomaly detection. We propose a new LSTM layer, the Attention Free Long Short-Term Memory (AF-LSTM), for volatility prediction. This new layer shows superior predictive performance compared to the original version. For anomaly detection, a conditional autoencoder (AF-CAE) is introduced, providing better anomaly detection, and reducing false positives. This thesis proposes several contributions in the field of time series prediction; the results demonstrate improvements in terms of accuracy and robustness, opening new perspectives for the modeling and management of financial assets.

Keywords: Deep Learning, Econometrics, Time Series, State Space Models, Forecasting.

Résumé

Cette thèse, structurée en quatre parties distinctes, contribue à enrichir les domaines de l'apprentissage profond et de l'économétrie non linéaire. Les modèles traditionnels ont souvent montré des faiblesses pour capturer les comportements non linéaires et les changements de régime caractéristiques des actifs numériques. Cette thèse se concentre sur l'amélioration de la prédiction des rendements et de la volatilité de cette nouvelle classe d'actifs à travers le développement de nouvelles architectures de réseaux de neurones et de méthodologies innovantes. Dans une première partie, nous introduisons les réseaux de Kolmogorov-Arnold temporels (TKAN) ainsi qu'une nouvelle architecture de type transformer (TKAT) reposant sur les TKAN pour la prévision des séries temporelles. Ces modèles intègrent la gestion de la mémoire et le mécanisme d'attention pour capturer les dynamiques complexes des marchés financiers. Nos résultats montrent que le TKAN et le TKAT surpassent les modèles récurrents traditionnels (LSTM, GRU) en termes de stabilité et de précision de prédiction, particulièrement sur des horizons de temps plus longs. La deuxième partie propose une nouvelle approche de modélisation des régimes de marché à l'aide de réseaux de neurones récurrents et des nouveaux modèles GARCH à changements de régimes Markoviens. La première approche permet de détecter les régimes de marché et d'estimer les probabilités de transition entre les différents états. Les résultats montrent une amélioration significative dans la détection et la prédiction des régimes de marché par rapport aux modèles traditionnels. Dans la seconde famille de modèles, de nouvelles spécifications de type MS-GARCH sont définies; des procédures d'inférence statistique par simulation sont proposées et testées. Dans la troisième partie, nous explorons l'utilisation des signatures pour la construction de portefeuilles. Une méthode de classification basée sur les signatures est développée pour sélectionner les actifs les plus représentatifs de chaque cluster, améliorant ainsi le ratio rendement-risque. Les portefeuilles construits avec cette méthodologie montrent des performances supérieures par rapport aux portefeuilles standards. La dernière partie de cette thèse aborde la modélisation de la volatilité et la détection d'anomalies. Nous proposons une nouvelle couche LSTM, l'Attention Free Long Short-Term Memory (AF-LSTM), pour la prédiction de la volatilité. Cette nouvelle couche montre des performances prédictives supérieures à la version originale. Pour la détection des anomalies, un auto-encodeur conditionnel (AF-CAE) est introduit, offrant une meilleure détection des anomalies et réduisant les faux positifs. Cette thèse propose plusieurs contributions dans le domaine de la prédiction des séries temporelles les résultats démontrent des améliorations en termes de précision et de robustesse, ouvrant de

nouvelles perspectives pour la modélisation et la gestion des actifs financiers.

Keywords: Apprentissage profond, économétrie, séries temporelles, modèles d'espace d'état, volatilité, prévisions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction (en Français)

La prédiction des séries temporelles est un domaine de recherche actif depuis de nombreuses années. De nombreux modèles ont été introduits dans la littérature, chacun présentant ses propres forces et faiblesses. L'objectif principal de cette recherche est de concevoir des modèles qui allient précision, efficacité et reproductibilité. Actuellement, le domaine de la recherche en finance s'active sur la modélisation des rendements d'une nouvelle classe d'actif : les cryptomonnaies, ainsi que sur leur volatilité et les volumes échangés. Cette nouvelle classe d'actif s'est démarquée par sa volatilité très élevée. Ses dernières années ont été rythmées par des phases de marché marquées par de fortes hausses, puis par de fortes baisses. La dynamique des rendements des cryptomonnaies n'est sûrement pas de type linéaire. En particulier, leurs prix ne varient pas suivant une marche aléatoire ou même un processus ARMA. Cette problématique peut être due notamment à la présence de différents régimes de marché.

Pour répondre à ce problème, certains modèles issus de l'économétrie non linéaire ont été introduits. Ces derniers permettent de prendre en compte les différents états de marchés existants. Avec le développement de l'apprentissage automatique et notamment les avancées dans le développement d'architectures de réseaux de neurones, de nouveaux horizons se sont ouverts pour la prévision de séries temporelles [Vas+17; Lim+21]. Ce chapitre introductif présente trois problèmes de prédiction en finance qui ont été traités dans cette thèse: la prédiction de la volatilité, des volumes et des rendements. Nous explorons de nouvelles frontières dans le domaine du deep learning en introduisant des architectures de réseaux de neurones innovantes. Dans la première partie de cette thèse, nous aborderons la temporalité et la gestion de la mémoire au sein des réseaux Kolmogorov-Arnold (KANs) [Liu+24]. Puis, dans une deuxième partie, nous proposons un nouveau cadre pour l'identification des régimes de marché présents dans les séries temporelles. Nous y intégrons le concept "d'espace-état" dans les réseaux neuronaux profonds. Cette approche, peu explorée jusqu'à présent, est pleine de promesses concernant notre compréhension et notre utilisation des modèles de deep learning, en offrant une structure plus dynamique, flexible et qui serait adaptée à certains actifs très volatils tels que les cryptomonnaies. Dans la troisième partie, nous proposons l'utilisation de "signatures" pour la construction de portefeuilles. Nous y présentons également une nouvelle architecture KANs qui repose sur cet objet mathématique. Enfin, dans la quatrième partie de cette thèse, nous introduisons une

architecture de réseau neuronal inédite conçue spécifiquement pour la modélisation de la volatilité. Cette architecture vise à capturer de manière plus efficace les fluctuations et les incertitudes inhérentes aux données, un défi persistant dans de nombreux domaines d'application. L'objectif de cette introduction sera de mettre en exergue les éléments-clés des parties sus-décrites. Elle évoquera également le contexte de recherche et les enjeux qui ont motivé la spécification de nouveaux modèles.

1 Part I: Les réseaux Kolmogorov-Arnold (KANs) pour les séries temporelles

1.1 Contexte

Récemment, une nouvelle architecture de réseau de neurones a été introduite, les Kolmogorov-Arnold Networks (KANs) [Liu+24]. Cette architecture est une alternative au Multi Layer Perceptrons (MLP) [Cyb89], et se fonde sur le théorème de représentation de Kolmogorov-Arnold [Kol61]. Le théorème de représentation de Kolmogorov-Arnold stipule que toute fonction continue multivariée peut être représentée comme la composition de sommes de fonctions univariées. Dans cette partie, nous avons proposé deux nouvelles architectures basées sur les KANs, mais avec une caractéristique innovante : la capacité de traiter des données séquentielles, non présente dans le modèle initial. Dans un premier temps, nous avons introduit le Temporal Kolmogorov Arnold Networks (TKAN). Ensuite, nous avons proposé une architecture basée sur le mécanisme d'attention, le Temporal Kolmogorov Arnold Transformer (TKAT). Pour ces deux modèles, nous avons utilisé la même tâche d'apprentissage et de test, pour mesurer au mieux la performance et l'intérêt des extensions proposées.

1.2 Chapitre 2: TKAN: Temporal Kolmogorov Arnold Networks

Motivation

Les réseaux neuronaux récurrents (RNN) ont révolutionné de nombreux domaines, en particulier le traitement du langage naturel et le traitement des séquences de données. Les réseaux avec mémoire à long terme (LSTM) ont démontré leur capacité à capturer les dépendances de long terme dans les données séquentielles. Inspirée par les réseaux de Kolmogorov-Arnold (KAN), et alternative prometteuse aux perceptrons multicouches (MLP), notre nouvelle architecture de réseaux neuronaux combine KAN et LSTM; elle est appelée "réseaux de Kolmogorov-Arnold temporels" (TKANs). Ces réseaux sont composés de couches de réseaux de Kolmogorov-Arnold récurrents (RKAN) qui intègrent la gestion de la mémoire. Cette innovation nous permet d'effectuer des prévisions de séries temporelles en plusieurs étapes avec une précision et une efficacité accrues. En s'attaquant aux limites des modèles traditionnels dans le traitement des modèles séquentiels complexes, l'architecture TKAN, quant à elle, offre un potentiel significatif d'avancées dans les domaines nécessitant des prévisions à plusieurs horizons temporels.

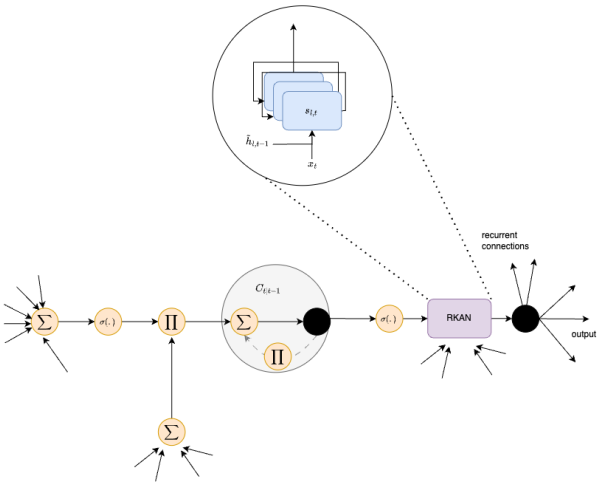


Figure 1.1 – TKAN Layer

Temporal Kolmogorov-Arnold Networks (TKAN)

Dans le Chapitre 2, nous proposons une nouvelle architecture conçue pour maintenir la mémoire des entrées passées. Cette architecture intègre les états cachés précédents dans les états actuels, ce qui permet au réseau de présenter un comportement temporel dynamique. En d'autres termes, cette approche permet au réseau de se souvenir et de prendre en compte les informations des entrées antérieures lors du traitement des nouvelles données, améliorant ainsi la capacité du réseau à gérer des séquences temporelles et à répondre de manière plus cohérente et informée aux variations des entrées. Le noyau récurrent est essentiel pour permettre aux couches RKAN d'apprendre à partir de séquences où le contexte et l'ordre jouent un rôle crucial. Nous avons conçu le TKAN pour tirer parti de la puissance du réseau Kolmogorov-Arnold tout en offrant une gestion de la mémoire pour traiter la dépendance temporelle. Les résultats pour des tâches de prédiction sont très prometteurs, spécialement pour la prédiction sur plusieurs horizons temporels, comme le démontrent les tableaux ci-dessous. Pour évaluer les performances du modèle, nous répétons l'expérience 5 fois pour chacun, et affichons ci-dessous la moyenne et l'écart-type des résultats d'entraînement obtenus de ces 5 expériences.

1.2.1 Résumé des métriques de performance

Table 1.1 – Moyenne (R^2) obtenue sur 5 exécutions

Temps	TKAN:5 B-Spline	GRU:default	LSTM:default	Dernière Valeur
1	0.33736	0.365136	0.355532	0.292171
3	0.21227	0.200674	0.061220	-0.062813
6	0.13784	0.082504	-0.225838	-0.331346
9	0.09803	0.087164	-0.290584	-0.457718
12	0.10401	0.017864	-0.473220	-0.518252
15	0.09512	0.033423	-0.404432	-0.555633

Table 1.2 – Écart-type R^2 obtenu sur 5 exécutions

Temps	TKAN:5 B-Spline	GRU:default	LSTM:default
1	0.00704	0.008336	0.011163
3	0.00446	0.004848	0.080200
6	0.01249	0.023637	0.062710
9	0.02430	0.014833	0.052729
12	0.00132	0.086386	0.085746
15	0.00701	0.024078	0.092729

Les résultats montrent que le coefficient de détermination (R^2) tend à diminuer à mesure que l'horizon de temps de prédiction augmente. Ceci est logique, plus l'horizon de temps de prédiction est long, plus il est difficile de prédire avec précision. Cependant, les résultats montrent aussi deux choses. Tout d'abord, bien que tous les modèles aient une différence de performance relativement faible sur un horizon court tel que 1 à 3 périodes, les performances changent beaucoup plus avec un horizon de temps plus long. Le modèle LSTM devient même inutile et contre-productif avec 6 périodes, tandis que TKAN et GRU obtiennent toujours une valeur R^2 moyenne plus élevée. Le TKAN se distingue avec des horizons de temps plus longs, avec une valeur R^2 au moins 25% supérieure à celle de GRU. Enfin, il faut étudier la stabilité du modèle, c'est-à-dire la capacité à calibrer des poids à partir d'échantillons sans trop de variation d'une expérience à l'autre. Ici encore, le TKAN a montré une stabilité bien meilleure que tous les autres modèles.

Au Chapitre 2 vous trouverez la présentation détaillée du modèle.

1.3 Chapitre 3: A Temporal Kolmogorov-Arnold Transformer for Time Series Forecasting

Motivation

La prévision de séries temporelles multivariées est un domaine d'intérêt majeur pour les chercheurs [TSB10; TT89; Wei18; MSW06] et est devenue un domaine de recherche important pour de nombreuses industries. Le volume de données disponibles augmente; il est donc naturel de proposer des cadres de prédiction plus avancés. Contrairement aux séries temporelles univariées, dans lesquelles une seule variable est analysée au fil du temps, les séries temporelles multivariées impliquent plusieurs variables interdépendantes. Ces connexions rendent la tâche d'apprentissage plus complexe. Les données de séries temporelles multivariées nécessitent des modèles sophistiqués capables de capturer les interactions dynamiques et les dépendances temporelles sur plusieurs dimensions et horizons temporels. Les chercheurs ont exploré diverses méthodes, y compris les modèles autorégressifs vectoriels [ZW06; Lüt13], les modèles à espace d'états [Ham94; Kim94] ou encore les réseaux de neurones [TDF91], pour relever les défis associés à la prévision de séries temporelles multivariées [BMD18; ML19]. Les approches d'apprentissage profond, telles que les réseaux Long Short-Term Memory (LSTM) et les mécanismes d'attention, ont montré des résultats prometteurs en raison de leur capacité à gérer les non-linéarités et les dépendances à long terme dans les données [HS97b; Vas+17]. Ces techniques avancées sont cruciales pour améliorer la précision des prévisions et prendre des décisions éclairées basées sur les résultats prédits.

1.4 Temporal Kolmogorov-Arnold Transformer (TKAT)

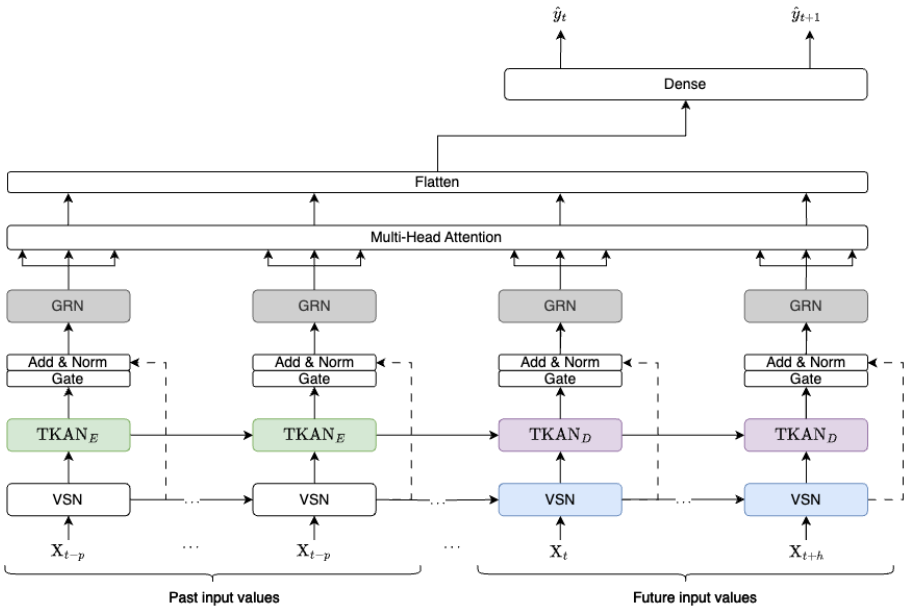


Figure 1.2 – TKAT Architecture

Dans ce chapitre, nous avons proposé une nouvelle architecture Transformer utilisant les couches du Temporal Kolmogorov-Arnold Networks (TKAN). L'idée est d'abord d'utiliser des couches TKAN au lieu de couches LSTM pour l'encodage et le décodage. Puis, de proposer une architecture adaptée aux cas d'usage où les entrées connues/déterministes ne sont pas majoritaires tandis que les entrées observées le sont. Ce sont des cas d'usage typiques dans le secteur de la finance. Il s'agit d'une différence majeure par rapport à la tâche standard pour laquelle le TFT [Lim+21] a été conçu. Nous avons observé ici que, pour les séries avec peu d'entrées connues, notre architecture offre de bien meilleures performances. Les résultats obtenus pour le TKAT montrent une puissance prédictive plus faible par rapport à des modèles plus simples dans le cas d'une prédiction pour une période de temps. Cependant, dès que le modèle est utilisé pour prédire à 3 périodes de temps successives, il offre de meilleures performances que tous les autres modèles de référence. Les résultats montrent que les performances obtenues sont plus stables au cours de l'exécution et sur le nombre de périodes, avec un coefficient de détermination (R^2) plus de 50% supérieur à celui du modèle TKAN. Une autre observation intéressante est la comparaison entre le modèle utilisant la couche TKAN et celui basé sur les LSTM. Bien que l'architecture du modèle permette aux LSTM de performer beaucoup mieux que seuls, il apparaît que la couche TKAN est capable d'améliorer les performances d'environ 5% en moyenne.

Table 1.3 – R^2 moyen : TKAT vs références

Temps	TKAT	TKAT non-KAN	TKAN	GRU	LSTM
1	0,30519	0,29834	0,33736	0,36513	0,35553
3	0,21801	0,22146	0,21227	0,20067	0,06122
6	0,17955	0,17584	0,13784	0,08250	-0,22583
9	0,16476	0,15378	0,09803	0,08716	-0,29058
12	0,14908	0,15179	0,10401	0,01786	-0,47322
15	0,14504	0,12658	0,09512	0,03342	-0,40443

Table 1.4 – Écart-type de R^2 : TKAT vs références

Temps	TKAT	TKAT non-KAN	TKAN	GRU	LSTM
1	0,01886	0,01610	0,00704	0,00833	0,01116
3	0,00906	0,00485	0,00446	0,00484	0,08020
6	0,00654	0,00352	0,01249	0,02363	0,06271
9	0,00896	0,00578	0,02430	0,01483	0,05272
12	0,00477	0,00507	0,00132	0,08638	0,08574
15	0,01014	0,01106	0,00701	0,02407	0,09272

Les résultats montrent que l'utilisation de l'architecture complète conduirait à de moins bons résultats pour les tâches spécifiques étudiées. L'architecture proposée est donc clairement différente de celle de TFT, mais n'est probablement pas adaptée aux mêmes tâches d'apprentissage

par construction. Bien que l'inspiration de TFT donne d'excellents résultats pour améliorer les performances de modèles simples, elle doit être adaptée à des tâches d'apprentissage spécifiques, car ici les résultats montrent non seulement une moyenne plus faible pour le R^2, mais aussi une bien plus grande variance dans les résultats.

Table 1.5 – R^2 moyen : variantes de TKAT plus proches de TFT

Temps	TKAT-A	TKATN-A	TKAT-B	TKATN-B
1	0,29237	0,29502	0,28861	0,31119
3	0,19352	0,11464	0,16413	0,12965
6	0,11667	0,11588	0,14149	0,12075
9	0,07502	0,10003	0,09655	0,05756
12	0,05745	0,06382	0,07805	0,05968
15	0,04640	0,03845	0,05999	0,04139

Table 1.6 – Écart-type de R^2 : variantes de TKAT plus proches de TFT

Temps	TKAT-A	TKATN-A	TKAT-B	TKATN-B
1	0,04861	0,03480	0,08500	0,00833
3	0,02206	0,04145	0,02807	0,02665
6	0,02927	0,02070	0,01188	0,01567
9	0,01558	0,02216	0,01776	0,01475
12	0,01437	0,01354	0,01431	0,02162
15	0,01191	0,00834	0,01612	0,00937

1.5 Nombre de paramètres par modèle

Le modèle TKAT présente près de 10 fois plus de paramètres que les modèles de référence en raison de son architecture plus complexe. Cependant, près de 65% de ces paramètres appartiennent à la partie de sélection des variables du réseau, tandis que les couches TKAN/LSTM ne représentent que 10% du total des poids entraînables.

Table 1.7 – Nombre de paramètres entraînables par modèle

Temps	TKAT	TKAT non-KAN	TKAN	GRU	LSTM
1	1,022,275	1,061,461	99,426	97,001	128,501
3	1,027,077	1,066,263	99,628	97,203	128,703
6	1,043,280	1,082,466	99,931	97,506	129,006
9	1,070,283	1,109,469	100,234	97,809	129,309
12	1,108,086	1,147,272	100,537	98,112	129,612
15	1,156,689	1,195,875	100,840	98,415	129,915

1.6 Résultats sur la tâche de prévision à 30 pas

Enfin, les résultats de l'erreur quadratique moyenne, pour chaque période de temps en fonction du modèle, montrent que, bien que le GRU soit meilleur dans le cas d'un horizon de prévision, le TKAT et le TKAN améliorent la qualité de la prédiction lorsque le nombre d'horizons augmente.

Table 1.8 – Nombre de paramètres entraînables

Modèle	Nombre de paramètres entraînables	R^2
TKAT	1,561,704	0,127743
TKAN	102,355	0,068004
MLP	298,230	-0,036020
GRU	99,930	0,055263
LSTM	131,430	-0,008245

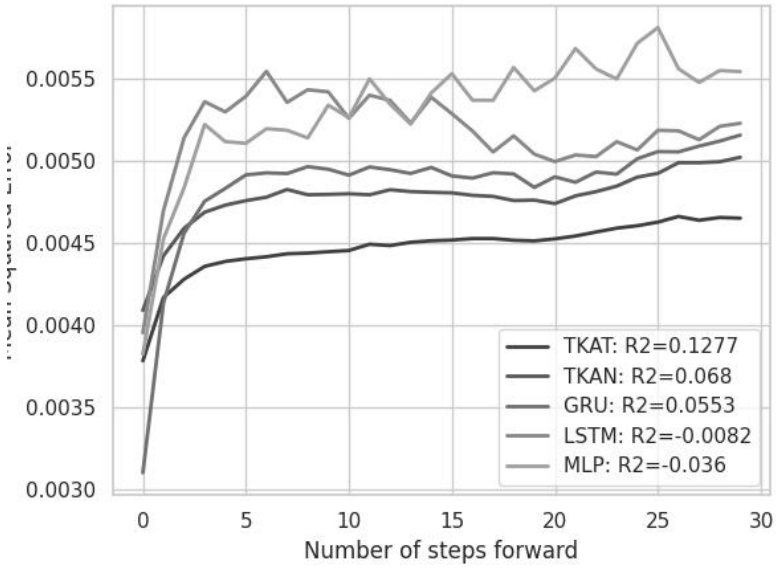


Figure 1.3 – Model and errors based on number of steps forward

La Figure 1.3 a été obtenue à partir d'un "single run". Elle montre clairement la nette surperformance du GRU sur la première période, tout comme les résultats obtenus précédemment. Mais, on observe également que plus le nombre de périodes de prévisions augmente, plus le TKAT et TKAN ont une capacité de prédiction supérieure aux autres modèles.

Au Chapitre 3 vous trouverez une présentation détaillée du TKAT.

2 Part II: Exploration des modèles à changement d'états pour les séries temporelles

2.1 Contexte

Nakamoto [Bit08] a créé la première cryptomonnaie décentralisée basée sur la technologie blockchain, le bitcoin. Sa vocation initiale était de créer un système de paiement peer-to-peer. Cependant, les utilisateurs du bitcoin l'utilisent principalement comme un actif de spéculation [Gla+14], et non comme un moyen de paiement, contrairement à sa conception originale. Le marché des actifs numériques est un marché "event-driven"; cela signifie que les prix de ces actifs peuvent fluctuer de manière significative en réponse à des événements ou des nouvelles diffusées par la presse ou les réseaux sociaux. Ces événements peuvent avoir un impact positif ou négatif, et peuvent être liés directement à une cryptomonnaie ou à son écosystème. Au cours de cette dernière année, la conjoncture économique et l'environnement réglementaire ont rythmé les cryptomonnaies. Une annonce majeure, telle que l'adoption d'une cryptomonnaie par une grande institution financière ou la commercialisation d'un nouveau véhicule d'investissement institutionnel comme les ETFs (Exchange Traded Funds), peut entraîner une forte hausse du prix de la cryptomonnaie. À l'inverse, une réglementation plus stricte ou une cyber-attaque peut provoquer une forte baisse du prix. Ces événements peuvent déclencher des pics ("spikes") de volatilité où des rendements extrêmes peuvent être observés; dans d'autres cas, ils peuvent être à l'origine de changements brusques de régime de marché, comme nous l'avons vu avec la chute de la plateforme d'échanges FTX.com. La volatilité du marché des cryptomonnaies est une autre caractéristique qui contribue à son caractère "event-driven". Les prix des cryptomonnaies peuvent fluctuer de manière significative en peu de temps, rendant la prédiction des mouvements de prix difficile. Le marché des cryptomonnaies a été marqué par de fortes hausses et baisses au cours de cette dernière décennie, indiquant la présence de régimes de marché distincts. Cette dynamique motive notre innovation, articulée autour de deux axes principaux : l'introduction d'un nouveau modèle MS GARCH et la proposition d'une nouvelle architecture de réseaux de neurones pour identifier les régimes de marché.

2.2 Chapitre 5: A new view on Markov-Switching Garch

Motivation

Depuis l'introduction des modèles à changements de régime pour l'analyse économétrique des cycles économiques par Hamilton [Ham89a], cette famille de modèles a été largement étudiée dans la littérature académique et parmi les praticiens. Ces modèles reposent sur l'existence de plusieurs états du marché, et les transitions entre ces états sont aléatoires et sont régies par une chaîne de Markov discrète et non observable (latente). Dans de tels modèles paramétriques, les paramètres varient dans le temps, et leur valeur à un instant donné t est conditionnée par l'état s_t de la chaîne à cette même date. Lorsque la chaîne de Markov est homogène (situation typique), la loi de cette chaîne de Markov est déduite d'une matrice de transition notée M et

de dimension de $m \times m$ telle que

$$M = [p_{ij}],\ p_{i,j} = \mathbb{P}(s_t = j | s_{t-1} = i),\ (i,j)\ \in 1, \ldots, m^2. \tag{1.1}$$

L'application des modèles à changement de régimes markoviens dans le domaine de l'économétrie financière a émergé comme une idée fructueuse. L'analyse empirique des rendements des actifs financiers affiche des traits empiriques distincts et remarquables : des queues épaisses, une asymétrie, et surtout, une agrégation de volatilité en "clusters". Cette dernière caractéristique étant traditionnellement prise en compte par des modèles de type GARCH, certains auteurs ont proposé de les étendre sous forme de modèles Markov-Switching GARCH (MS-GARCH) [HMP04; BPR10], pour combiner l'idée de l'existence de différents régimes et la persistance des volatilités des rendements. La spécification la plus naturelle du modèle est alors la suivante : supposons qu'une série de rendements financiers $(r_t)_{t=1,\ldots,T}$ suit la dynamique suivante:

$$r_t = \mu_{s_t} + \sigma_t \epsilon_t, \tag{1.2}$$

où l'innovation notée (ϵ_t) est une séquence de variables aléatoires indépendantes et identiquement distribuées. (s_t) suit une chaîne de Markov discrète et indépendante de (ϵ_t). Supposons qu'il existe m états sous-jacents, le processus de la volatilité conditionnelle s'écrit de la manière suivante:

$$\sigma_t^2 = \omega_k + a_k z_{t-1}^2 + b_k \sigma_{t-1}^2,\ \text{quand}\ s_t = k,\ k \in \{1, \ldots, m\}, \tag{1.3}$$

où z_t désigne une information de marché disponible à la date t. Selon les auteurs, il est supposé que $z_t = r_t - \mu_{s_t}$, $z_t = r_t - \mathbb{E}_{t-1}[\mu_{s_t}]$, ou même $z_t = r_t$ simplement.

L'équation (1.3) décrit le processus GARCH(1,1), mais des modèles bien plus complexes ont été introduits par la suite. Ces modèles ont permis de tenir compte de l'asymétrie, de barrières, etc. Toutefois, malgré les extensions proposées, la variance conditionnelle σ_t^2 ne dépend pas du régime à la date t mais des valeurs précédentes jusqu'à la date $t-1$ rendant alors le calcul de la vraisemblance numériquement difficile. Les premières tentatives pour résoudre ce problème ont été faites par [Gra96] et [Kla02].Une approche plus commode a été proposée par [HMP04]: ils supposent l'existence de m dynamiques parallèles

$$\sigma_{k,t}^2 = \omega_k + a_k r_{t-1}^2 + b_k \sigma_{k,t-1}^2,\ \text{avec}\ k \in \{1, \ldots, m\}, \tag{1.4}$$

où le processus du rendement de l'actif est $r_t := \sigma_{s_t,t}\epsilon_t$. L'espérance conditionnelle de r_t étant supposée nulle, μ_{s_t} ne figure donc pas dans la spécification du rendement. Presque tous les auteurs qui se sont intéressés aux modèles MS-GARCH supposent cette moyenne conditionnelle μ_{s_t} nulle pour des raisons pratiques. Il est toutefois relativement simple de modifier la méthodologie de [HMP04] en présumant $r_t := \mu_{s_t} + \sigma_{s_t,t}\epsilon_t$. La volatilité conditionnelle dans ce cas s'écrirait de la manière suivante:

$$\sigma_{k,t}^2 = \omega_k + a_k (r_{t-1} - \mu_k)^2 + b_k \sigma_{k,t-1}^2,\ \text{pour}\quad \text{tout}\ \ k \in \{1, \ldots, m\}. \tag{1.5}$$

Une nouvelle approche

Pour en revenir à l'équation (1.4), il apparaît que le modèle de [HMP04], même s'il est pratique (notamment à des fins d'inférence), souffre de certaines faiblesses en termes d'interprétabilité et ne soutient pas bien l'intuition financière. Il est alors difficile d'identifier la rationalité derrière les spécifications proposées des équations (1.4)-(1.5). Pour mieux représenter la coexistence de plusieurs dynamiques de volatilité, nous suggérons d'apporter des modifications aux équations mentionnées précédemment, ce qui donnerait alors

$$r_t = \mu_{s_t} + \sigma_{s_t,t}\epsilon_t, \text{ et} \tag{1.6}$$

$$\sigma^2_{k,t} = \omega_k + a_k\sigma^2_{k,t-1}\epsilon^2_{t-1} + b_k\sigma^2_{k,t-1}, \text{ for any } k \in \{1,\dots,m\}. \tag{1.7}$$

Le modèle que nous proposons consiste donc à supposer que l'équation d'actualisation de la volatilité conditionnelle est désormais

$$r_t = \mu_{s_t} + \sigma_{s_t,t}\epsilon_t, \text{ et} \tag{1.8}$$

$$\sigma^2_{k,t} = \omega_k + a_k\sigma^2_{k,t-1}\epsilon^2_{t-1} + b_k\sigma^2_{k,t-1}, \text{ pour tout } k \in \{1,\dots,m\}. \tag{1.9}$$

Dans cette nouvelle spécification équation (1.9), (ϵ_t) le vecteur d'innovation, est utilisé pour updater tous les processus de volatilité. En présumant l'existence de dynamiques parallèles, ce choix peut être discuté. En effet, plusieurs processus d'innovation différents pourraient être considérés en fonction des régimes. Pour des états agités, on pourrait faire le choix d'une autre distribution que la loi normale pour introduire de l'asymétrie sur notre vecteur d'innovation.

Dans cette nouvelle spécification du modèle - équation (1.9) -, le vecteur d'innovation (ϵ_t) est utilisé pour mettre à jour l'ensemble des processus de volatilité. En tenant compte de l'existence de dynamiques de volatilité en parallèle, l'utilisation de ce vecteur peut susciter des interrogations. En effet, il serait envisageable d'adopter différents processus d'innovation en fonction des différents régimes. Pour des périodes plus turbulentes, il pourrait être judicieux de sélectionner une distribution autre que la loi normale $\mathcal{N}(0,1)$ pour les innovations, afin d'introduire une asymétrie dans le vecteur d'innovation. Cette réflexion nous pousse alors à proposer une alternative à notre première proposition équation (1.9). Un autre nouveau modèle pourrait alors être:

$$r_t = \mu_{s_t} + \sigma_{s_t,t}\epsilon_{s_t,t}, \tag{1.10}$$

l'équation "d'update" de la volatilité deviendra alors

$$\sigma^2_{k,t} = \omega_k + a_k\sigma^2_{k,t-1}\epsilon^2_{k,t-1} + b_k\sigma^2_{k,t-1}, \text{ pour tout } k \in \{1,\dots,m\}, \tag{1.11}$$

avec notre vecteur d'innovation $\vec{\epsilon_t} := (\epsilon_{1,t},\dots,\epsilon_{m,t})$. Sous cette spécification, il y aurait donc de nombreuses quantités aléatoires cachées par rapport à (1.4) ou même (5.7). L'inférence de (1.10)-(1.11) sera alors difficile en raison du nombre potentiellement élevé de variables latentes.

Le détails des différents modèles proposés sont fournis au Chapitre 5.

2.3 Chapitre 4: Deep State Space Models for Time Series Forcasting

Cette sous-section présente le premier sujet que nous allons traiter dans cette thèse, vous trouverez plus d'information au Chapitre 4.

Motivation

Les rendements financiers sont souvent caractérisés par des dépendances non linéaires complexes et des comportements de changement de régime imprévisibles. Ces particularités rendent la modélisation complexe, posent des défis d'analyse et de compréhension approfondie des processus stochastiques sous-jacents. Dans le Chapitre 4, nous proposons une nouvelle solution pour la modélisation en introduisant de nouvelles architectures de réseaux de neurones pour la détection de régime de marché dans les séries temporelles. Nous proposons une nouvelle architecture de réseau de neurones basés sur des RNNs, permettant alors la présence de plusieurs régimes et d'estimer la probabilité d'être dans un régime à une date donnée. Cette architecture est également capable de prendre en compte le fait que la probabilité d'être dans un régime peut changer au cours du temps grâce à sa capacité à traiter les données temporelles.

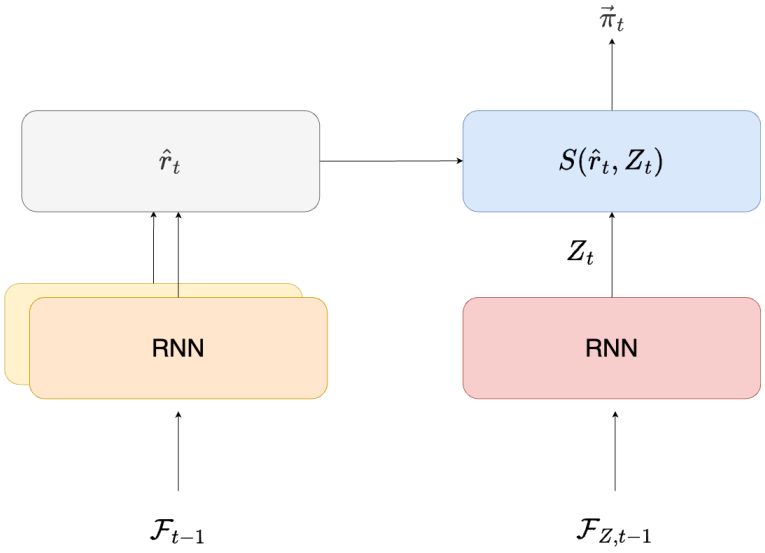


Figure 1.4 – m-RNN

Sur la partie gauche du réseau détaillé à la Figure 1.4, on voit plusieurs réseaux de type RNNs; on notera chacun d'entre eux $\text{RNN}_{k,in}$. Ces RNNs sont entraînés en parallèle. Chacun d'entre eux représente un régime de marché. Sur le côté droit de la Figure 1.4, nous avons un autre RNN qui sera noté RNN_c, et sera alimenté uniquement par les covariables (non par l'ensemble des données contenant les covariables et les valeurs observées passées des rendements). RNNc transformera les covariables en un vecteur Z_t tel que

$$Z_t = \text{RNN}c(\mathcal{F}_{Z,t-1}).$$

Les Z_t et le vecteur $\vec{r_t} = (r_{1,t}, r_{2,t}, \ldots, r_{m,t})$ obtenus précédemment avec l'utilisation de RNNk, in seront les entrées de notre mécanisme de changement de régime détaillé en 4.1 pour estimer le vecteur $\vec{\pi_t} = (\pi_{1,t}, \pi_{2,t}, \ldots, \pi_{m,t})$. La sortie du cadre est $\vec{\pi_t}$. Dans notre modèle à changement de régime, la fonction $S(r_t, Z_t)$ génère un vecteur de probabilités π_t pour m régimes à l'instant t. Pour un système à deux régimes, $\pi_t = (\pi_{t,1}, \pi_{t,2})$. Le régime qui prédit $\hat{y}_t$ est déterminé par $\hat{s}_t = \arg\max_k(\pi_{t,k})$, ce qui peut être exprimé par une fonction indicatrice $I_{t,k}$. Le régime réel $s_t \in \{0, 1\}$ est codé en one-hot. La matrice de confusion C est alors construite comme $C_{ij} = \sum_t I(\hat{s}_t = i \text{ et } s_t = j)$, où $\hat{s}_t$ est le régime prédi et s_t ce que nous avons défini comme le régime réel.

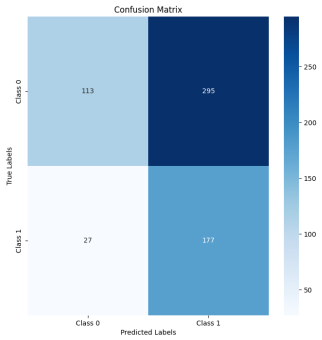


Figure 1.5 – GRU Confusion matrix (out of sample)

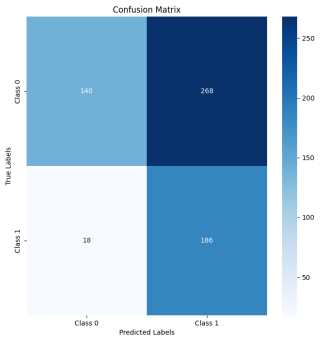


Figure 1.6 – LSTM Confusion matrix (out of sample)

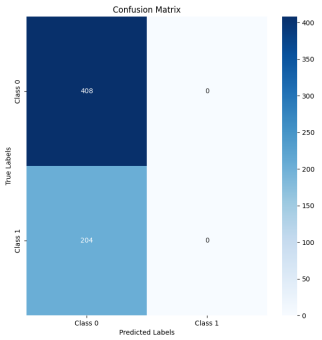


Figure 1.7 – TKAN Confusion matrix (out of sample)

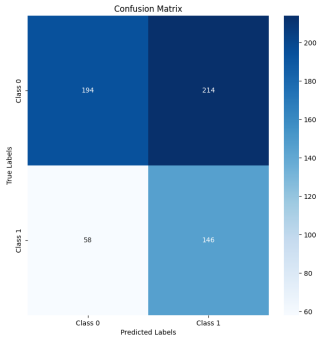


Figure 1.8 – m-GRU Confusion matrix (out of sample)

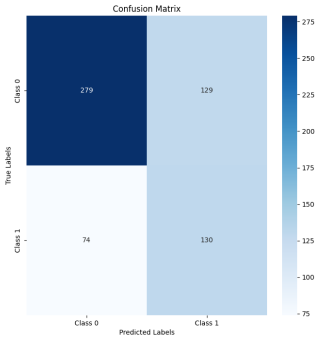


Figure 1.9 – m-LSTM Confusion matrix (out of sample)

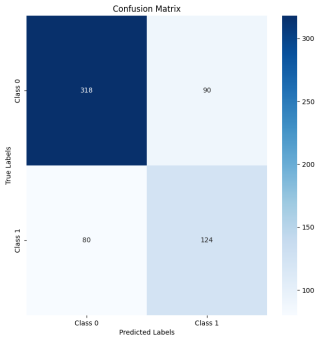


Figure 1.10 – m-TKAN Confusion matrix (out of sample)

Table 1.9 – Comparison of simple RNNs versus Switching RNNs

Model	**Class**	**Precision**	**Recall**	**F1-Score**	**Support**	**Accuracy**
LSTM vs. m-LSTM						
LSTM	Class 0	**0.89**	0.34	0.49	408	0.53
	Class 1	0.41	**0.91**	0.57	204	
m-LSTM	Class 0	0.79	**0.68**	**0.73**	408	**0.67**
	Class 1	**0.50**	0.64	**0.56**	204	
GRU vs. m-GRU						
GRU	Class 0	**0.81**	0.28	0.41	408	0.47
	Class 1	0.38	0.87	0.52	204	
m-GRU	Class 0	0.77	**0.48**	**0.59**	408	**0.56**
	Class 1	**0.41**	**0.72**	**0.52**	204	
TKAN vs. m-TKAN						
TKAN	Class 0	0.67	**1.00**	**0.80**	408	0.67
	Class 1	0.00	0.00	0.00	204	
m-TKAN	Class 0	**0.79**	0.78	0.79	408	**0.72**
	Class 1	**0.58**	**0.59**	**0.58**	204	

Le tableau ci-dessus montre que notre architecture à chaîne de Markov améliore significativement la capacité du modèle à apprendre et à prédire des régimes. En effet, les RNNs "classiques" conventionnels, comme les GRU ou les LSTM, sont incapables de discerner les nuances d'un régime, comme en témoigne la performance supérieure du TKAN, qui identifie simplement la classe dominante. Cependant, nos modèles à chaîne de Markov démontrent une précision améliorée dans les prévisions pour tous les modèles. Malgré la qualité des modèles basés sur GRU, ceux intégrant des unités TKAN présentent une précision supérieure notable. Le modèle basé sur TKAN, en particulier, démontre des performances robustes sur des tâches "out of sample".

Au Chapitre 4 vous trouverez le détail de ces modèles.

3 Part III: Apprentissage à partir des signatures

3.1 Contexte

Comme évoqué précédemment, les cryptomonnaies se distinguent des autres classes d'actifs par leur très forte volatilité. Plusieurs chercheurs ont étudié le comportement et la modélisation des processus latents pour déterminer les risques inhérents aux crypto-actifs. ([BM+16; BD17; PK17], etc.) Récemment, le clustering basé sur la corrélation a été utilisé pour déduire les connexions hiérarchiques entre différents actifs en analysant la matrice de corrélation de leurs rendements. Néanmoins, aucun consensus n'a émergé concernant un modèle fiable pour prédire le prix de ces nouveaux actifs. En effet, les facteurs influençant les prix des cryptomonnaies sont nombreux, complexes et difficiles à prendre en compte dans leur intégralité. Ils peuvent être d'origines diverses: économiques, réglementaires ou encore techniques [Kri15]. La particularité de ces nouveaux actifs soulève de nouveaux enjeux en gestion de portefeuille. Dans cette partie de la thèse, nous proposons, dans un premier chapitre, une nouvelle approche pour la gestion de portefeuille avec un procédé innovant sur la construction d'univers d'investissement; puis, dans un second chapitre, nous proposons une nouvelle architecture de type Kolmogorov-Arnold Networks pondérée par des signatures.

3.2 Chapitre 6: Clustering Digital Assets Using Path Signatures

Cette sous-section présente la construction de portefeuille composé de cryptomonnaies. Les détails de méthodologies sont détaillés dans le Chapitre 6.

Motivation

Dans ce chapitre, notre principale motivation est de proposer une approche novatrice pour la construction de portefeuilles. L'innovation se trouve au coeur de la construction d'un univers d'investissement, dans lequel différents portefeuilles optimaux seront construits et backtestés. Nous cherchons à créer cet univers d'investissement composé d'actifs numériques, à l'aide des signatures [Che58]. Une signature est un objet mathématique qui résume de manière succincte les propriétés fondamentales d'un path. L'approche par signatures a récemment été adoptée dans de nombreux domaines, tels que l'apprentissage automatique [CK16; Per+18; Fer21]), l'analyse de séries temporelles ([Gyu+13; DCS21]) ou la vision par ordinateur ([Yan+22; LZJ17]). On peut définir une signature de la manière suivante: définissons un path de N-dimension tel que $(X_t)_{t\in[0,T]}$. En d'autres termes, $X_t = (Xt^1, ..., X_t^N)$. Les "paths" seront désignés par (X_t^n), $n \in 1, \ldots, N$. Pour construire itérativement la signature d'un "path" (X_t), nous considérons d'abord les incréments de $X^1, \ldots, X^N$ sur n'importe quel intervalle $[0, t]$, $t \in [0, T]$. Ils sont désignés $S(X)^1_{0,t}, \ldots, S(X)^N_{0,t}$ et définis comme suit:

$$S(X)^n_{0,t} := \int_0^t dX_s^n. \tag{1.12}$$

$S(X)^n_{0,t}$ est la première étape pour calculer la signature d'un "path" unidimensionnel. C'est une séquence de nombres réels, chacun de ces nombres correspondant à une intégrale itérée du "path". Ensuite, les coefficients de signature suivants impliqueront deux "paths": un "path" de coordonnées (X^m_t) et le "path" d'incrément $(S(X)^n_{0,t})$, associé au "path" de coordonnées (X^n_t). Il existe N^2 intégrales de second ordre qui sont désignées $S(X)^{1,1}_{0,t} \dots, S(X)^{N,N}_{0,t}$, où

$$S(X)^{n,m}_{0,t} := \int_0^t S(X)^n_{0,s}, dX^m_s, ; n, m \in 1, \dots, N. \tag{1.13}$$

Ces signatures serviront d'input à notre procédure de construction de portefeuille. L'idée dans ce chapitre est de proposer une nouvelle méthodologie qui va permettre de réduire le nombre de trades d'une stratégie tout en améliorant sa performance. Cette approche est décrite dans figure 1.11.

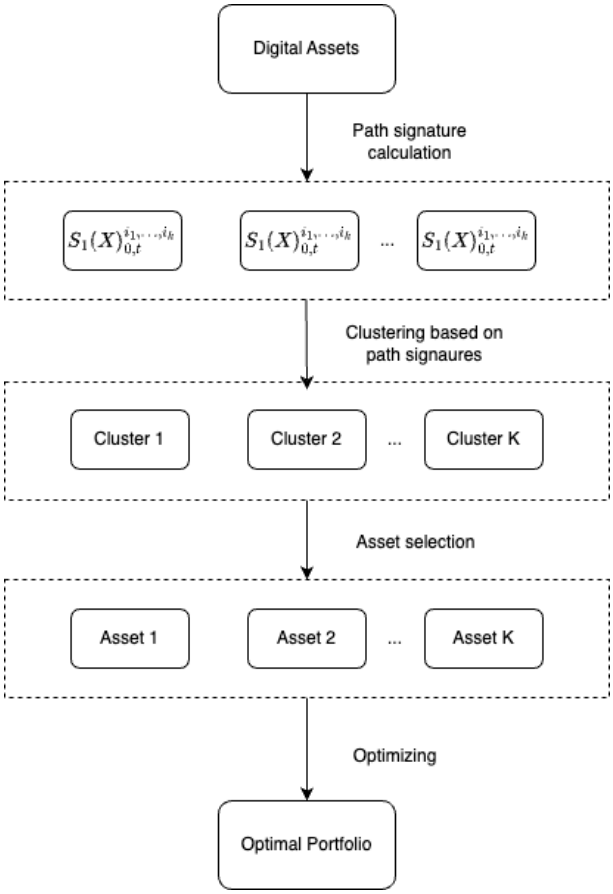


Figure 1.11 – Méthodologie de construction de portefeuille

Notre objectif est de présenter une nouvelle méthodologie visant à améliorer le ratio rendement-risque en sélectionnant les actifs les plus représentatifs de chaque "cluster". Ces "clusters" seront estimés à partir des signatures des actifs numériques. Une fois que l'univers d'investissement est défini, nous procédons au backtesting de chaque portefeuille pour évaluer son impact sur la performance et la gestion du risque. Les Figures 6.22-6.23 montrent le nombre de transactions pour chaque méthodologie, en utilisant la fenêtre Fixe de Temps (FOT) et la fenêtre glissante

(RW). Sur les deux figures, nous pouvons clairement identifier les stratégies les moins coûteuses en termes de frais de transaction. Pour notre backtest, nous avons utilisé des frais de transaction de 20 points de base. Étant donné que les gestionnaires d'actifs peuvent utiliser ces stratégies, nous n'avons pas envisagé de frais de transaction fixes pour chaque transaction, principalement en raison de l'irréalisme que cela suggère. Les résultats pour chacune des simulations sont communiqués dans le Chapitre 6 de cette thèse.

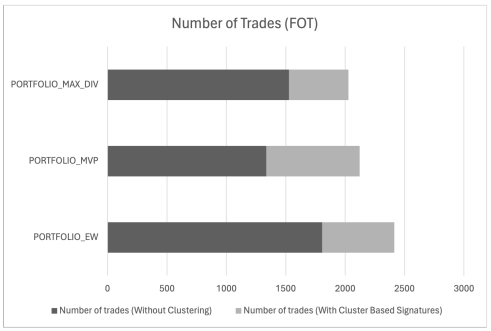


Figure 1.12 – Number of trades (FOT)

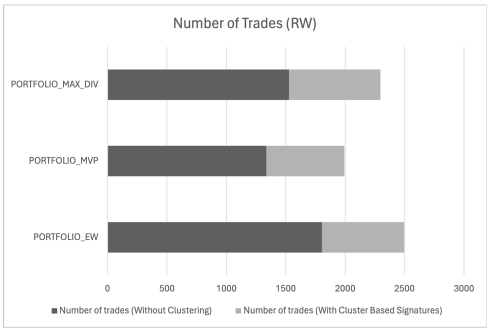


Figure 1.13 – Number of trades (RW)

Dans ce chapitre, nous avons introduit une nouvelle méthode de classification des actifs financiers, qui diffère des approches traditionnelles centrées sur les rendements historiques. Nous avons choisi d'exploiter les signatures de chemin, offrant une alternative robuste pour synthétiser l'information contenue dans les prix des actifs numériques. Cette méthode nous permet de découvrir une représentation différente, facilitant l'identification de similitudes entre différents actifs. Elle nous permet également de travailler dans un espace de dimension supérieure, tout en tenant compte de la dépendance temporelle. Les résultats obtenus avec différents portefeuilles démontrent clairement les avantages de cette approche, notamment pour les investisseurs. Pour le portefeuille (EW), la version filtrée par regroupement, désignée (EW_{sc}^{FOT}) et (EW_{sc}^{RW}), surpasse nettement la méthodologie standard (non filtrée) en termes de rendements annualisés (0,9592, 1,2523 contre 0,5984) et de volatilité annualisée (0,6319, 0,7432 contre 0,8219). Cette performance supérieure est également soutenue par des ratios de Sharpe et de Calmar plus élevés, et un DrawDown Maximum (MDD) plus faible, indiquant un rendement ajusté au risque plus efficace et une diminution moindre de la valeur au fil du temps. Une analyse similaire avec les portefeuilles Mean-Variance montre que le MVP filtré par regroupement (MVP_{sc}^{FOT}) affiche également un rendement annualisé plus élevé (0,2199 contre 0,1140) par rapport au MVP standard (MVP). Cependant, (MVP_{sc}^{RW}) rapporte une faible performance de 0,1488 pour un niveau de risque équivalent à celui de (MVP_{sc}^{FOT}). En ce qui concerne les mesures de risque, il y a des rendements ajustés au risque légèrement meilleurs pour (MVP_{sc}^{FOT}), mais atténués par un MDD plus élevé. Enfin, la comparaison du MDP révèle que les MDP filtrés par regroupement, désignés (MDP_{sc}^{FOT}) et (MDP_{sc}^{RW}), génèrent un rendement annualisé plus élevé (1,1903, 0,4013 contre 0,2543) que le MDP standard. Le coût du rendement plus élevé est le risque du portefeuille, les versions filtrées par regroupement affichant une volatilité annualisée plus élevée (0,7603, 0,6676 contre 0,5742). Cependant, les mesures de risque indiquent des rendements ajustés au risque meilleurs pour (MDP_{sc}^{RW}), bien qu'il subisse un MDD plus élevé.

Dans l'ensemble, les portefeuilles filtrés par regroupement démontrent systématiquement des rendements plus élevés par rapport à leurs homologues standards, à l'exception du (MVP). Cependant, ces avantages sont accompagnés d'une volatilité et d'un MDD accrus, indiquant un compromis entre des rendements plus élevés et des risques croissants. Les investisseurs doivent peser ces facteurs avec soin, en tenant compte de leur tolérance au risque et de leurs objectifs d'investissement.

3.3 Chapitre 7: Signature-Weighted Kolmogorov-Arnold Networks (KANs) for Time Series

Motivation

Dans ce dernier chapitre, nous proposons une nouvelle architecture de réseaux de neurones, les Signature-Weighted Kolmogorov-Arnold Networks (SigKAN). La motivation pour ce modèle est d'intégrer aux couches KAN une représentation complémentaire lors de l'apprentissage, représentation qui est obtenue en utilisant les signatures sur les observations. L'avantage clé est la possibilité de tirer parti de la puissance des signatures tout en conservant la structure modulaire et les bonnes propriétés d'approximation des réseaux KAN [Liu+24]. Cela en fait une architecture intéressante pour résoudre des problèmes d'apprentissage impliquant des données séquentielles, comme dans le secteur de la finance.

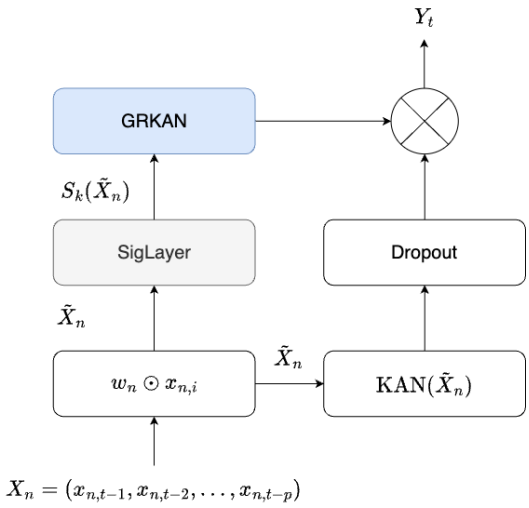


Figure 1.14 – SigKAN Architecture

La Figure 1.14 décrit l'architecture du SigKAN; on peut voir qu'elle est composée de signatures apprenables (SigLayer) ainsi que d'un Gated Residual KAN (GRKAN). On voit également que la transformation d'un "path" pour le rendre apprenable permet d'alimenter le SigLayer et un KAN. Le GRKAN, qui prend en entrée une signature apprenable, produira en sortie un vecteur de probabilités. Ce vecteur de probabilités pondérera les valeurs obtenues des couches

KAN notées KAN($\hat{X}_n$) pour ajuster la prédiction.

Les résultats se concentrent sur deux indicateurs des modèles SigKAN et SigDense : la performance en termes de R^2 moyen et l'écart-type de R^2 sur plusieurs exécutions pour évaluer la stabilité des modèles. Les tableaux ci-dessous résument les performances des modèles SigKAN, SigDense et des benchmarks.

Time	SK-1	SD-1	SK-2	SD-2
1	0.36225	0.33055	0.30807	0.31907
3	0.21961	0.21532	0.20580	0.21066
6	0.16361	0.15544	0.15351	0.15836
9	0.13997	0.12768	0.12684	0.13338
12	0.12693	0.11628	0.11826	0.11814
15	0.11861	0.11097	0.11448	0.11065

Table 1.10 – R^2 Moyen pour SigKAN et SigDense

Time	TKAT	TKAN	GRU	LSTM
1	0.30519	0.33736	0.36513	0.35553
3	0.21801	0.21227	0.20067	0.06122
6	0.17955	0.13784	0.08250	-0.22583
9	0.16476	0.09803	0.08716	-0.29058
12	0.14908	0.10401	0.01786	-0.47322
15	0.14504	0.09512	0.03342	-0.40443

Table 1.11 – R^2 Moyen pour les benchmarks

Les modèles SigKAN surpassent les modèles de référence (TKAN, GRU, LSTM) en termes de R^2 moyen, en particulier pour les prédictions à court terme. Toutefois, ils n'atteignent pas les performances du TKAT pour les périodes de prédiction plus longues. L'utilisation de la signature comme mécanisme de pondération est efficace, et les couches KAN améliorent les performances par rapport aux couches denses.

Le modèle SigKAN montre des résultats plus stables à travers les différentes exécutions, notamment pour les tâches de prédiction à plus long horizon, comparé aux modèles basés sur les réseaux de neurones récurrents. Cette stabilité accrue pourrait être due à l'élimination de couches "récurrentes". Le détails des résultat est disponible au Chapitre 7 de cette thèse.

4 Part IV: Les séries temporelles ont besoin d'attention

4.1 Contexte

La volatilité et les rendements des cryptomonnaies constituent des sujets de recherche de premier plan en raison de leur nature unique et de leur comportement bien souvent imprévisible. Contrairement aux actifs financiers traditionnels, les cryptomonnaies sont caractérisées par des volatilités élevées et des rendements potentiellement élevés. Cette dynamique particulière s'explique par divers facteurs, notamment la liquidité, le sentiment de marché, la spéculation et les annonces médiatiques [Gla+14]. La volatilité, définie comme l'écart-type des rendements, mesure l'amplitude des variations du prix d'un actif. Plus la volatilité est élevée, plus le risque pour un investisseur est élevé, et inversement. Cette caractéristique rend la volatilité cruciale dans le processus de prise de décision des investisseurs, influençant leurs choix d'investissement ainsi que les méthodes de gestion des risques opérées par les banques et autres institutions financières [ABR19]. Le marché des cryptomonnaies est souvent décrit comme un marché "event-driven", où les prix peuvent fluctuer de manière significative en réponse à des événements spécifiques ou des nouvelles diffusées par la presse et les réseaux sociaux. Ces événements peuvent inclure l'adoption de cryptomonnaies par des institutions financières, des changements réglementaires ou des cyber-attaques, chacun pouvant provoquer des mouvements de prix rapides et imprévisibles [Bit08]. Traditionnellement, les modèles linéaires ont été utilisés pour prédire la volatilité et les rendements des actifs financiers. Cependant, ces modèles se révèlent insuffisants pour capturer la complexité et la nature non linéaire des dynamiques de marché des cryptomonnaies.

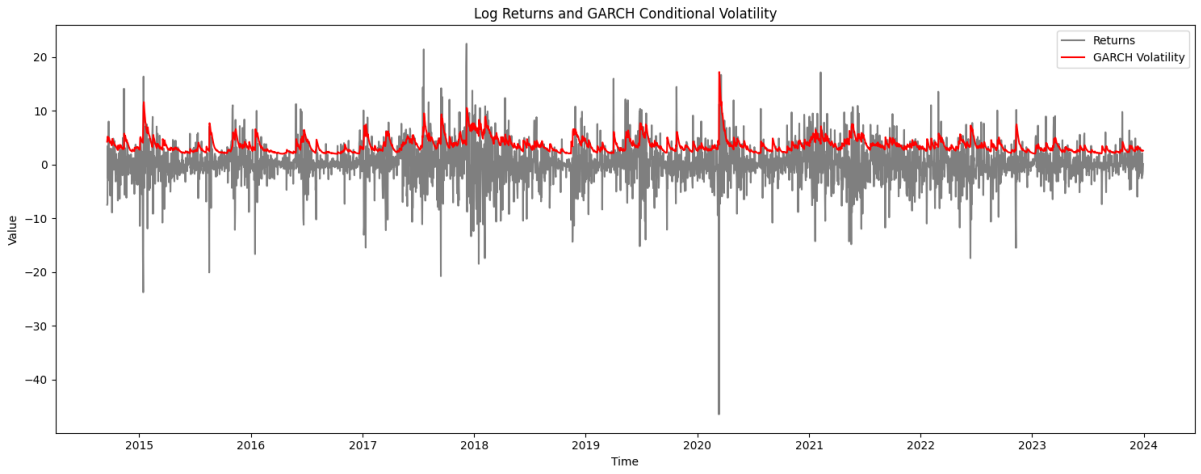


Figure 1.15 – Log rendements vs volatilité conditionelle

Pour répondre à ces défis, nous proposons deux innovations dans cette dernière partie, en intégrant des mécanismes d'attention dans la prévision des séries temporelles complexes. Les mécanismes d'attention permettent de se concentrer sur les aspects les plus pertinents des données historiques, améliorant ainsi la précision des prédictions et offrant ainsi une plus-value significative dans l'analyse des rendements et de la volatilité.

Dans un premier chapitre, nous introduisons une nouvelle couche de réseau de neurones, l'AF-LSTM (Attention Free Long Short-Term Memory) une extension du LSTM, pour améliorer la prédiction de la volatilité. Ce modèle combine les rendements au carré et les volatilités conditionnelles des modèles GARCH pour capturer les dynamiques complexes et non linéaires des marchés des actifs numériques. L'AF-LSTM utilise des mécanismes d'attention pour se concentrer sur les aspects les plus pertinents des séries temporelles financières, améliorant ainsi la qualité des prévisions. Dans un second chapitre, qui constituera le dernier chapitre de cette thèse, nous proposons un AF Conditional Autoencoder (Attention Free Conditional Autoencoder) conçu pour détecter les anomalies sur les marchés des actifs numériques. Cet Autoencodeur conditionnel utilise des mécanismes d'attention pour identifier les anomalies potentielles en fonction des conditions de marché actuelles et passées.

4.2 Chapitre 8 An Attention Free Long Short-Term Memory For Time Series Forecasting

Motivation

Avec le besoin croissant d'améliorer la précision de la modélisation de la volatilité, les modèles GARCH ont été un sujet de préoccupation sérieux dans la finance et la littérature économique. Cependant, la relation entre les variances conditionnelles étant non linéaire, ces modèles ont trouvé leurs limites. Certains travaux de recherche se concentrant sur l'introduction de la non-linéarité dans ces modèles s'appuient sur les réseaux de neurones pour introduire de la non-linéarité dans les modèles dits classiques [NTS13; KM15]. Kim et Won [KW18] ont mélangé les réseaux de neurones récurrents (RNNs) avec plusieurs modèles de type GARCH pour introduire une non-linéarité [LS20]. Dans ce chapitre, nous cherchons à améliorer la capacité prédicative des modèles GARCH. Nous ne remettons pas en question la pertinence de cette famille de modèles mais nous essaierons plutôt de corriger la prédiction faite à chaque date à l'aide des RNNs.

Dans un premier temps nous procédons à l'estimation du modèle GARCH(1,1):

$$\sigma_t^2 = \omega + \alpha_1 r_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \tag{1.14}$$

où $r_t := \sigma_t \epsilon_t$. Nous récupérons de cette estimation les volatilités conditionnelles estimées pour chacune des dates. Ces données seront stockées dans un vecteur noté X_{t-1}, qui sera l'input de deux modèles que nous comparererons en termes de capacités prédictives: le LSTM, et le AF-LSTM. Notre vecteur d'entrée est donc noté $X_{t-1} \in \mathbb{R}^{L \times N}$, où L désigne la taille de la séquence d'entrée, et N est le nombre de "features". Le modèle est détaillé dans le Chapitre 8 de cette thèse.

Bitcoin

Les tableaux 8.1-8.2 montrent les performances de prédiction du LSTM et de l'AF-LSTM. L'objectif pour ces modèles est de prédire les valeurs des carrés des rendement du bitcoin. Le modèle LSTM présente des performances constantes sur différents horizon de temps. Le

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0058	0.0763	0.0652	11.2880
t+2	0.0055	0.0744	0.0634	10.9141
t+3	0.0056	0.0746	0.0637	11.0528

Table 1.12 – Forecast using LSTM (Bitcoin out of sample).

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0012	0.0349	0.0253	16.0520
t+2	0.0012	0.0352	0.0255	16.8245
t+3	0.0012	0.0352	0.0256	17.1902

Table 1.13 – Forecast using AF-LSTM (Bitcoin out of sample).

modèle LSTM présente des valeurs d'erreur quadratique moyenne (MSE) et d'erreur quadratique moyenne racine (RMSE) relativement faibles. En revanche, le modèle AF-LSTM semble avoir des valeurs d'erreur quadratique moyenne et d'erreur quadratique moyenne encore plus faibles. Cela indique une précision de prévision supérieure à celle du modèle LSTM. Nous avons fait des tests sur plusieurs actifs numériques. Les résultats sont disponibles dans l'annexe du chapitre 9. Dans l'ensemble, les deux modèles présentent des résultats prometteurs pour prédire les futures rendements au carré du bitcoin. Le modèle AF-LSTM présentant des avantages en termes de minimisation de l'ensemble des mesures d'erreur, cela indique une meilleure performance sur différents actifs.

4.3 Chapitre 9 : An Attention Free Conditional Autoencoder For Anomaly Detection in Cryptocurrencies

Cette sous-section présente le dernier sujet traité dans cette thèse, la détection d'anomalies dans les rendements des actifs numériques. Les détails de méthodologies sont détaillés dans le chapitre 10.

Motivation

Dans ce chapitre, nous cherchons à répondre à une problématique importante dans le domaine de la finance : l'identification d'anomalies dans les rendements. La recherche sur les rendements extrêmes et les anomalies des rendements est un domaine qui suscite beaucoup d'intérêt en finance [Sor09; MS06]. Comprendre ces phénomènes, les modéliser, les prédire et les gérer plus efficacement est essentiel. La détection des anomalies nécessite la spécification d'un modèle adéquat, ce qui est crucial pour la gestion des risques financiers. En identifiant correctement ces anomalies, les investisseurs ou gestionnaires de portefeuilles peuvent prendre les mesures nécessaires afin de mitiger ce risque. Pour identifier les anomalies dans les log rendements des actifs numériques, nous avons d'abord défini une anomalie comme une erreur de prédiction

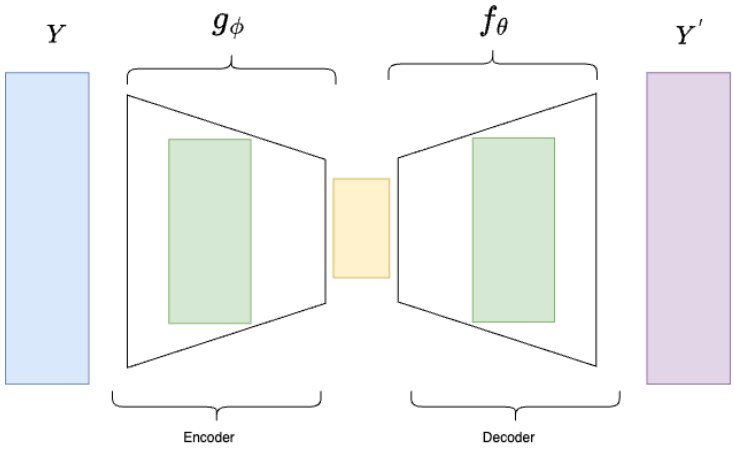


Figure 1.16 – Linear Encoder-Decoder

dont la valeur absolue dépasse le 95e centile des erreurs d'un échantillon, tout en ayant un log rendement en valeur absolue inférieur au 95e centile de ce même échantillon. Pour identifier les anomalies dans les séries temporelles, les auto-encodeurs se sont avérés très utiles [Pol+19; ZP17; SY14]. Un auto-encodeur est un type de réseau de neurones que l'on peut décomposer en deux étapes. La première, l'encodage des variables d'entrée, consiste en une compression des données. Puis, une deuxième étape consiste à décoder cette représentation des variables d'entrée compressée comme représenté dans la Figure 1.16. Mathématiquement, un auto-encodeur se définit de la manière suivante:

$$\begin{aligned} y_t &= f_\theta(g_\phi(y_t)) + u_t, \\ &= \theta^{(0)} + \theta^{(1)}(\phi^{(0)} + \phi^{(1)} y_t) + u_t, \end{aligned} \tag{1.15}$$

où $\phi := \{\phi^{(0)}, \phi^{(1)}\}$ et $\theta := \{\theta^{(0)}, \theta^{(1)}\}$. Dans le cas d'un auto-encodeur composé d'une seule couche, nos vecteurs de paramètres sont $\phi^{(1)}$, $\theta^{(1)}$, $\theta^{(0)}$ et $\phi^{(0)}$, respectivement. L'objectif de la tâche d'apprentissage est alors de minimiser u_t, l'erreur de reconstruction définie comme suit:

$$u_t = y_t - f_\theta(g_\phi(y_t)). \tag{1.16}$$

Nous proposons une alternative à cette technique afin d'affiner la détection d'anomalies en s'inspirant de la structure d'auto-encodeur conditionnel: c.f. Figure 1.17 proposé par Gu, Kelly, and Xiu [GKX21].

Structure Conditional Autoencoders

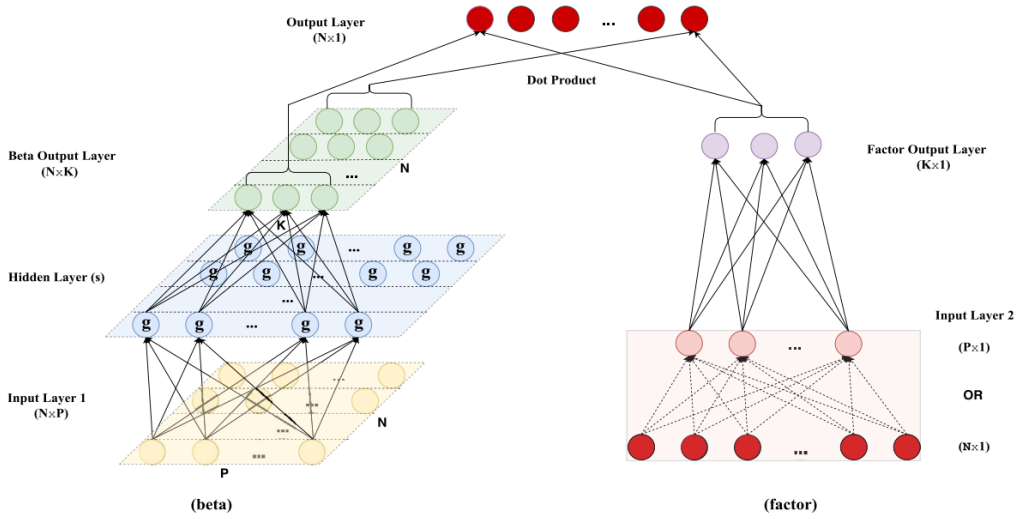


Figure 1.17 – Conditional Autoencoders (CAE) Structure [GKX21]

La Figure 1.17 illustre le modèle d'auto-encodeur conditionnel. La partie gauche de la figure représente les facteurs de chargement (factor loading) , qui sont des coefficients qui déterminent l'impact des caractéristiques de chaque cryptomonnaie. Ces facteurs de chargement sont dépendants des caractéristiques de la cryptomonnaie au temps t-1, qui sont représentées par le vecteur $Z_{t\text{-}1}$. La formulation récursive pour estimer les bétas s'écrit de la manière suivante:

$$\begin{aligned} z_{i,t-1}^{(0)} &= z_{i,t-1}, \\ z_{i,t-1}^{(l)} &= g(b^{(l-1)} + W^{(l-1)} z_{i,t-1}^{(l-1)}),\ l = 1, ..., l_\beta \\ \beta_{i,t-1} &= b^{(l_\beta)} + W^{(l_\beta)} z_{i,t-1}^{(l_\beta)}. \end{aligned} \tag{1.17}$$

La partie droite de la figure représente les facteurs inobservables, qui sont des facteurs qui influencent les rendements des actifs numériques, mais qui ne peuvent pas être directement observés. Ces facteurs inobservables sont représentés par le vecteur F_t.

Conditional Attention Free Autoencoder

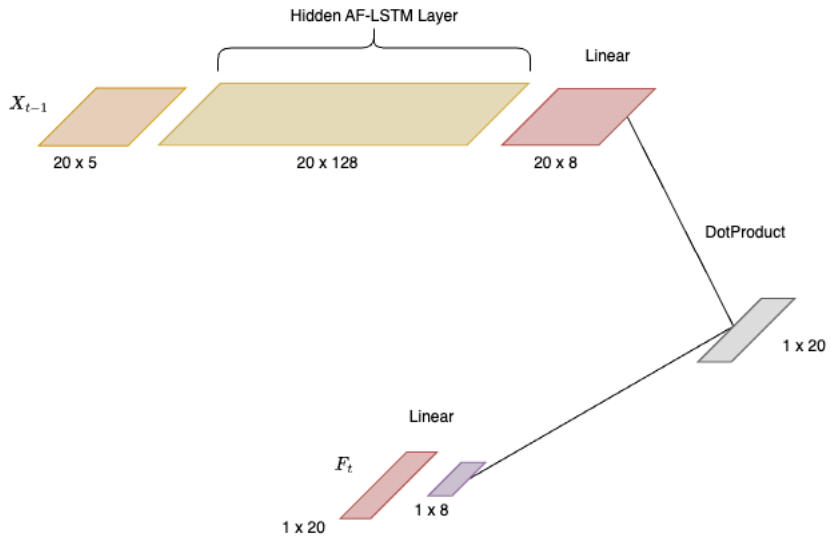


Figure 1.18 – Structure du Conditional Attention Free Autoencoder

Notre idée a alors été de conserver la même structure que celle proposée par Gu, Kelly, and Xiu [GKX21] mais d'intégrer un mécanisme d'attention particulier [Zha+21a], détaillé dans le chapitre 10. Cette innovation va permettre à notre auto-encodeur représenté sur la Figure1.18 de pouvoir se focaliser sur une partie de ces factor loadings, capables d'expliquer le rendement à la date t. Notre spécification du modèle permet de filtrer l'input X_{t-1} contenant les caractéristiques des cryptomonnaies.

	AF-LSTM CAE	LSTM CAE	Linear CAE
Loss	**0.4043**	0.5651	0.4961
R^2	**0.6277**	0.4811	0.5426

Table 1.14 – Metrics

Le Tableau 1.14 récapitule les différentes métriques (fonction de pertes, et R2) pour les 3 modèles que nous avons testés. Nous pouvons voir que le modèle AF-LSTM CAE affiche les meilleures performances prédictives, avec la plus faible fonction de perte de reconstruction et le plus haut coefficient de détermination. C'est un atout important pour la détection d'anomalies. En effet, l'AF-LSTM CAE est le plus performant pour reconstruire fidèlement les données d'entrée. Il sera également le plus à même d'identifier avec précision des erreurs de reconstruction significatives, définies comme celles supérieures au 95e centile en valeur absolue. Cette performance se traduira par une meilleure détection, avec moins de faux positifs identifiés à la Figure 1.19, qui seront plus nombreux sur les modèles LSTM CAE et Linear CAE. Cependant, la complexité de l'architecture à base d'attention du AF-LSTM CAE peut toutefois rendre

son interprétabilité et l'explicabilité plus complexes.

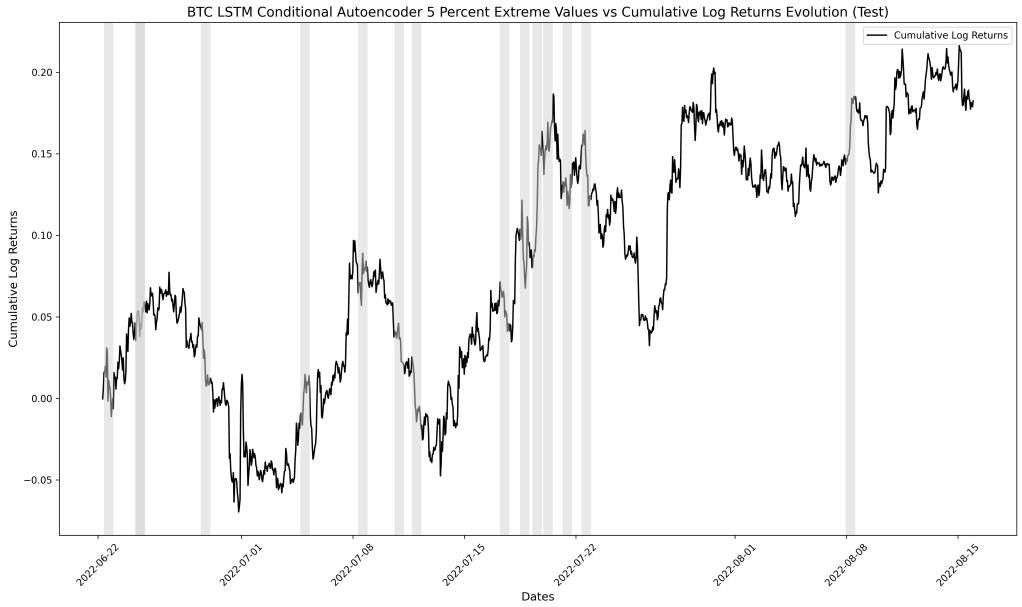


Figure 1.19 – BTC LSTM CAE 5 Percent Extreme Values Evolution on Test Set

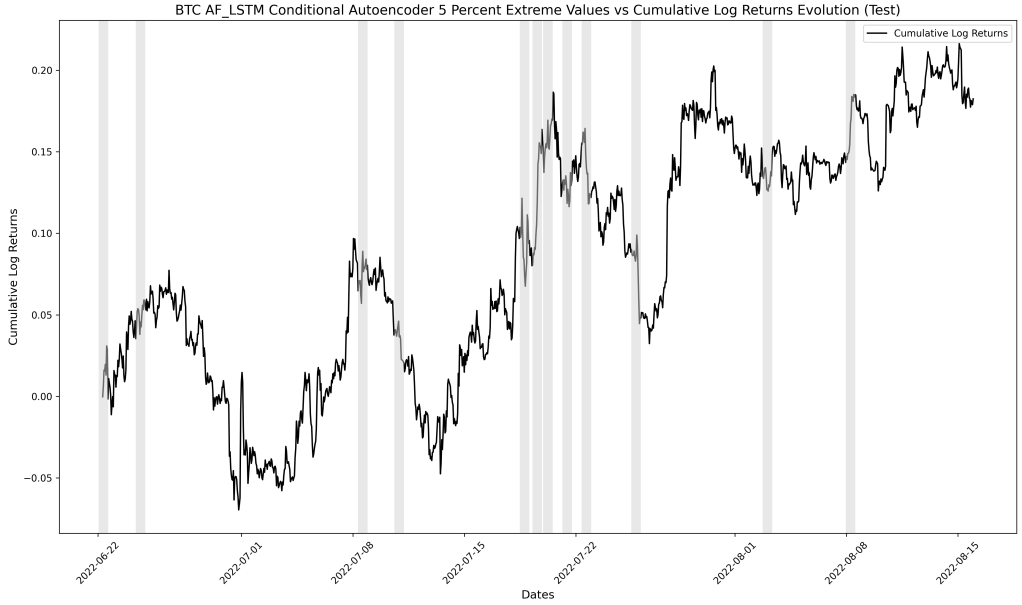


Figure 1.20 – BTC AF LSTM CAE 5 Percent Extreme Values Evolution on Test Set

Dans ce chapitre, nous cherchons donc à proposer un autoencoder avec un meilleur pouvoir explicatif pour affiner la détection d'anomalies. Notre modèle est détaillé dans le chapitre 10, dernier chapitre de cette thèse.

Part I

Kolmogorov-Arnold Networks (KANs) For Time Series

Chapter 2

TKAN: Temporal Kolmogorov Arnold Networks

This part is a joint work with Rémi Genet (Univ. Paris Dauphine).

Abstract

Recurrent Neural Networks (RNNs) have revolutionized many areas of machine learning, particularly in natural language and data sequence processing. Long Short-Term Memory (LSTM) networks have demonstrated their ability to capture long-term dependencies in sequential data. Inspired by the Kolmogorov-Arnold Networks (KANs), a promising alternatives to Multi-Layer Perceptrons (MLPs), we propose a new neural network architecture inspired by KAN and LSTM, called "Temporal Kolomogorov-Arnold Networks" (TKANs). TKANs combine the strenght of both networks. They are composed of Recurring Kolmogorov-Arnold Networks (RKANs) Layers embedding memory management. This innovation enables us to perform multi-step time series forecasting with enhanced accuracy and efficiency. By addressing the limitations of traditional models in handling complex sequential patterns, the TKAN architecture offers significant potential for advancements in fields requiring more than one step ahead forecasting.

Contents

1 Introduction

Time series forecasting is an important branch of statistical analysis and machine learning. The development of machine learning models has accelerated rapidly in the past few years. Time series, which can be defined as sequences of data indexed in time, are essential in finance, meteorology, and even in the field of healthcare. The ability to accurately predict the future evolution of such data has become a strategic issue for many different industries that are constantly seeking innovation. In recent years, the growing interest in this area of research is mainly due to the massive increase in the availability of data. Moreover, the increased computer processing capacity now makes it possible to process large datasets. As a second driver for innovation, new statistical methods, deep learning techniques [SGO20] and hybrid models have offered new opportunities to improve the accuracy and efficiency of forecasts. Besides econometric models such ARMA/ARIMA ([MH97; MSG14]) and their numerous extensions, Recurrent Neural Networks (RNNs) [MJ+01] yield families of models that have been recognized for their proven effectiveness in terms of forecasting [HBB21].

RNNs have been proposed to address the "persistence problem", i.e. the potential dependencies between the successive observations of some time series. Therefore, RNNs most often outperform "static" networks as MLPs [LJH15]. Traditional methods of gradient descent may not be sufficiently effective for training Recurrent Neural Networks (RNNs), particularly when it comes to capturing long-term dependencies [BSF94]. Meanwhile, [Chu+14] conducted an empirical study revealing the effectiveness of gated mechanisms in enhancing the learning capabilities of RNNs. Actually, RNNs have proved to be one of the most powerful tools for processing sequential data and solving a wide range of difficult problems in the fields of automatic natural language processing, translation, image processing and time series analysis where MLPs [HSW89; Hay98; Cyb89] cannot perform well. When it comes to sequential data management, MLPs face limitations. Unlike RNNs, MLPs, are not designed to manage sequential data; information only flows in one direction, from input to output (feedforward connection). This specification makes them not suitable for modeling temporal sequences where taking into account sequential patterns is essential for prediction. Another weakness of MLPs is that these networks have no embedded mechanism for cell memory state management. Recurrent Neural Networks (RNNs) [Wer90; Wil89] have provided an answer to these problems. However, standard RNNs have solved one problem, but created another: "vanishing" or "exploding gradient problem" [Hoc98b]. Traditional methods of gradient descent may not be sufficiently effective for training RNNs, particularly in capturing long-term dependencies [BSF94]. Meanwhile, [Chu+14] conducted an empirical study revealing the effectiveness of gated mechanisms in enhancing the learning capabilities of RNNs. Actually, RNNs have proved to be one of the most powerful tools for processing sequential data and solving a wide range of difficult problems in the fields of automatic natural language processing, translation and time series analysis. Recently, Liu et al. [Liu+24] proposed Kolmogorov-Arnold Networks (KANs) as an alternative to MLPs. The Kolmogorov-Arnold Network (KAN) is an efficient new neural network architecture known for its improved performance and interpretability. Unlike traditional models, KANs apply ac-

tivation functions on the connections between nodes, and these functions can even learn and adapt during training. In addition to using KAN Layer, Temporal Kolmogorov-Arnold Networks (TKANs) manage temporality within a data sequence. The idea is to design a new family of neural networks capable of catching long-term dependency. Our primary idea is to introduce an external memory module which can be attached to KAN Layers. This "memory" can store information that is relevant to the temporal context and can be accessed by the network during processing. This allows the network to explicitly learn and utilize past information. Codes are available at TKAN repository and can be installed using the following command: `pip install tkan`. Data are accessible if the reader wishes to reproduce our experiments using the GitHub link provided above.

2 Kolmogorov-Arnold Networks (KANs)

Multi-Layer Perceptrons (MLPs) [HSW89] are extension of original perceptron proposed by [Ros58]. They were inspired by the universal approximation theorem [Cyb89] which states that a feed-forward network (FFN) with a single hidden layer containing a finite number of neurons can arbitrarily well approximate any continuous functions on a compact subset of $\mathbb{R}^n$. On the other side, Kolmogorov-Arnold Networks (KAN) focuses on the Kolmogorov-Arnold representation theorem [Kol61]. The Kolmogorov-Arnold representation theorem states that any multivariate continuous function f can be represented as a composition of univariate functions and through additive operations:

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right) \tag{2.1}$$

where $\phi_{q,p}$ are univariate functions that map each input variable $x_p \in [0, 1]$ to $\mathbb{R}$, and $\Phi_q : \mathbb{R} \to \mathbb{R}$. Since all functions to be learnt are univariate, we can parametrize every 1D function as a B-spline curve. The learnable coefficients are then associated with local B-spline basis functions. The key insight comes when we see the similarities between MLPs and KAN. In MLPs, a layer includes a linear transformation followed by nonlinear operations, and you can make the network deeper by adding more layers. A KAN layer is rather

$$\mathbf{\Phi} = \{\phi_{q,p}\}, \quad p = 1, 2, \ldots, n_{\text{in}}, \quad q = 1, 2 \ldots, n_{\text{out}}, \tag{2.2}$$

where $\phi_{q,p}$ are parametrized function of learnable parameters. In the Kolmogorov-Arnold theorem, the inner functions form a KAN layer with $n_{\text{in}} = n$ and $n_{\text{out}} = 2n + 1$, and the outer functions form a KAN layer with $n_{\text{in}} = 2n + 1$ and $n_{\text{out}} = 1$. So, the Kolmogorov-Arnold representations in eq. (2.1) are simply compositions of two KAN layers. Now it becomes clear what it means to have Deep Kolmogorov-Arnold representation. Taking the notation from [Liu+24] let us define a shape of KAN $[n_0, n_1, \cdots, n_L]$, where n_i is the number of nodes in the i^{th} layer of the computational graph. We denote the i^{th} neuron in the l^{th} layer by (l, i), and the activation value of the (l, i)-neuron by $x_{l,i}$. Between layer l and layer $l + 1$, there are $n_l n_{l+1}$ activation

functions: the activation function that connects (l, i) and $(l+1, j)$ is denoted by

$$\phi_{l,j,i}, \quad l = 0, \cdots, L-1, \quad i = 1, \cdots, n_l, \quad j = 1, \cdots, n_{l+1}. \tag{2.3}$$

The pre-activation of $\phi_{l,j,i}$ is simply $x_{l,i}$; the post-activation of $\phi_{l,j,i}$ is denoted by $\tilde{x}_{l,j,i} \equiv \phi_{l,j,i}(x_{l,i})$. The activation value of the $(l+1, j)$ neuron is simply the sum of all incoming post-activations:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}), \qquad j = 1, \cdots, n_{l+1}. \tag{2.4}$$

Rewriting it under the matrix form will give:

$$\mathbf{x}_{l+1} = \underbrace{\begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}}_{\mathbf{\Phi}_l} \mathbf{x}_l, \tag{2.5}$$

where $\mathbf{\Phi}_l$ is the function matrix corresponding to the l^{th} KAN layer. A general KAN network is a composition of L layers: given an input vector $\mathbf{x}_0 \in \mathbb{R}^{n_0}$, the output of KAN is:

$$\text{KAN}(\mathbf{x}) = (\mathbf{\Phi}_{L-1} \circ \mathbf{\Phi}_{L-2} \circ \cdots \circ \mathbf{\Phi}_1 \circ \mathbf{\Phi}_0)\mathbf{x}. \tag{2.6}$$

3 Temporal Kolmogorov-Arnold Networks (TKANs)

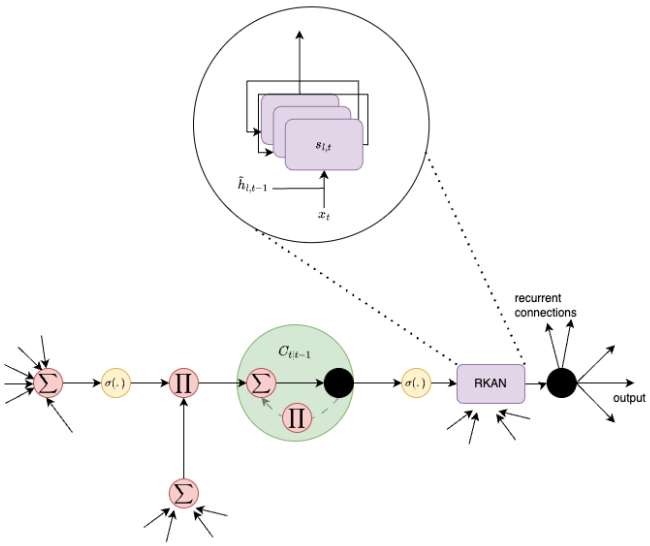


Figure 2.1 – Temporal Kolmogorov-Arnold Networks (TKAN)

After proposing the RKAN (Recurrent Kolmogorov-Arnold Network), which integrates temporality management by adapting the concept of Kolmogorov-Arnold networks to temporal sequences, we developed an additional innovation to build our neural network: the TKAN (Temporal Kolmogorov-Arnold Networks) layer. This TKAN layer combines the RKAN architecture with a slightly modified LSTM (Long Short-Term Memory) cell. The idea is to build an extension of the model proposed by [Liu+24] to manage sequential data and temporality during the learning task. The objective is to provide a framework for time series forecasting on multiple steps ahead. As discussed previously, RNNs' weakness generally lies in the difficulty of capturing persistence of information when some input sequence is quite long, resulting in a significant loss of information in some tasks. LSTMs address this problem through the use of a gating mechanism. LSTMs can be computationally more expensive than standard RNNs; however, this extra complexity is often justified by better performance on complex learning tasks. The integration of an LSTM cell combined with the RKAN enables the capture of complex nonlinearities with learnable activation functions of RKAN, but also the maintenance of a memory of past events over long periods with the LSTM cell architecture. This combination offers superior modeling power for tasks involving complex sequential data. The major components of the TKAN are:

— **RKAN Layers:** RKAN layers enable the retention of short term memory from previous states within the network. Each RKAN layer manages this short term memory throughout the processing in each layer.

— **Gating Mechanisms:** these mechanisms help to manage the information flow. The model decides which information should be retained or forgotten over time.

3.1 Recurring Kolmogorov-Arnold Networks (RKAN)

In deep learning, particularly in the context of recurrent neural networks (RNN), a recurrent kernel refers to the set of weights that are applied to the hidden state from the previous timestep during the network's operation. This kernel plays a crucial role in how an RNN processes sequential data over time. Let us denote $\tau = 1, 2, ...$ some discrete time steps. Each step has a forward pass and backward pass. During the forward pass, the output or activation of units are computed. During the backward pass, the computation of the error for all weights is made. During each timestep, an RNN receives an input vector and the hidden state from the previous timestep h_{t-1} . The recurrent kernel is a matrix of weights that transforms this previous hidden state. This operation is usually followed by an addition; the result of this transformation is passed through a non-linear function, an activation function $f(.)$, that can take many forms: tanh, ReLU, etc. The update stage could be formulated as:

$$h_t = f(W_{hh}h_{t-1} + W_{hx}x_t + b_h), \tag{2.7}$$

where h_t is the hidden state at time $t \in \tau$; W_h and W_x are the recurrent kernel, a weight matrix that transforms the previous hidden states h_{t-1}, and the input kernel, a weight matrix transforming the current input denoted x_t, respectively. In the next sections, we propose a new way of updating KANs. We propose a process to maintain the memory of past inputs by

incorporating previous hidden states into the current states, enabling the network to exhibit dynamic temporal behavior. Recurrent Kernel is the key so that RKAN layers learn from sequences for which context and order matter. We design the TKAN to leverage the power of Kolmogorov-Arnold Network while offering memory management to handle time dependency. To introduce time dependency, we modify each transformation function ϕ_l.

Let's denote the sublayers memory state $\tilde{h}_{l,t}$, initialized with zeros and of shape (KAN_{out}). The inputs that fed each sub KAN layers in the RKAN are created by doing:

$$s_{l,t} = W_{l,\tilde{x}} x_t + W_{l,\tilde{h}} \tilde{h}_{l,t-1}, \tag{2.8}$$

where $W_{l,\tilde{x}}$ is the weight of the l-th layer applied to x_t which is the input at time t. Moreover, $W_{l,\tilde{h}}$ are the weights of the l-th layer applied to its previous substate. Let us first denote KAN_{in} and KAN_{out}, the input dimension of RKAN Layer input and outputs, respectively. The shape of $W_{l,\tilde{x}} \in \mathbb{R}^{(d,\text{KAN}_{in})}$ and $W_{l,\tilde{h}} \in \mathbb{R}^{(\text{KAN}_{out},KAN_{in})}$, which leads to $s_{l,t}^{\text{KAN}_{in}}$. Set

$$\tilde{o}_t = \phi_l(s_{l,t}), \tag{2.9}$$

where ϕ_l is a KAN layer. The "memory" step $\tilde{h}_{l,t}$ is defined as a combination of past hidden states, such as:

$$\tilde{h}_{l,t} = W_{hh} \tilde{h}_{l,t-1} + W_{hz} \tilde{o}_t, \tag{2.10}$$

introducing vectors of weights, that weight the importance of past values relative to the most recent inputs.

3.2 TKAN Architecture

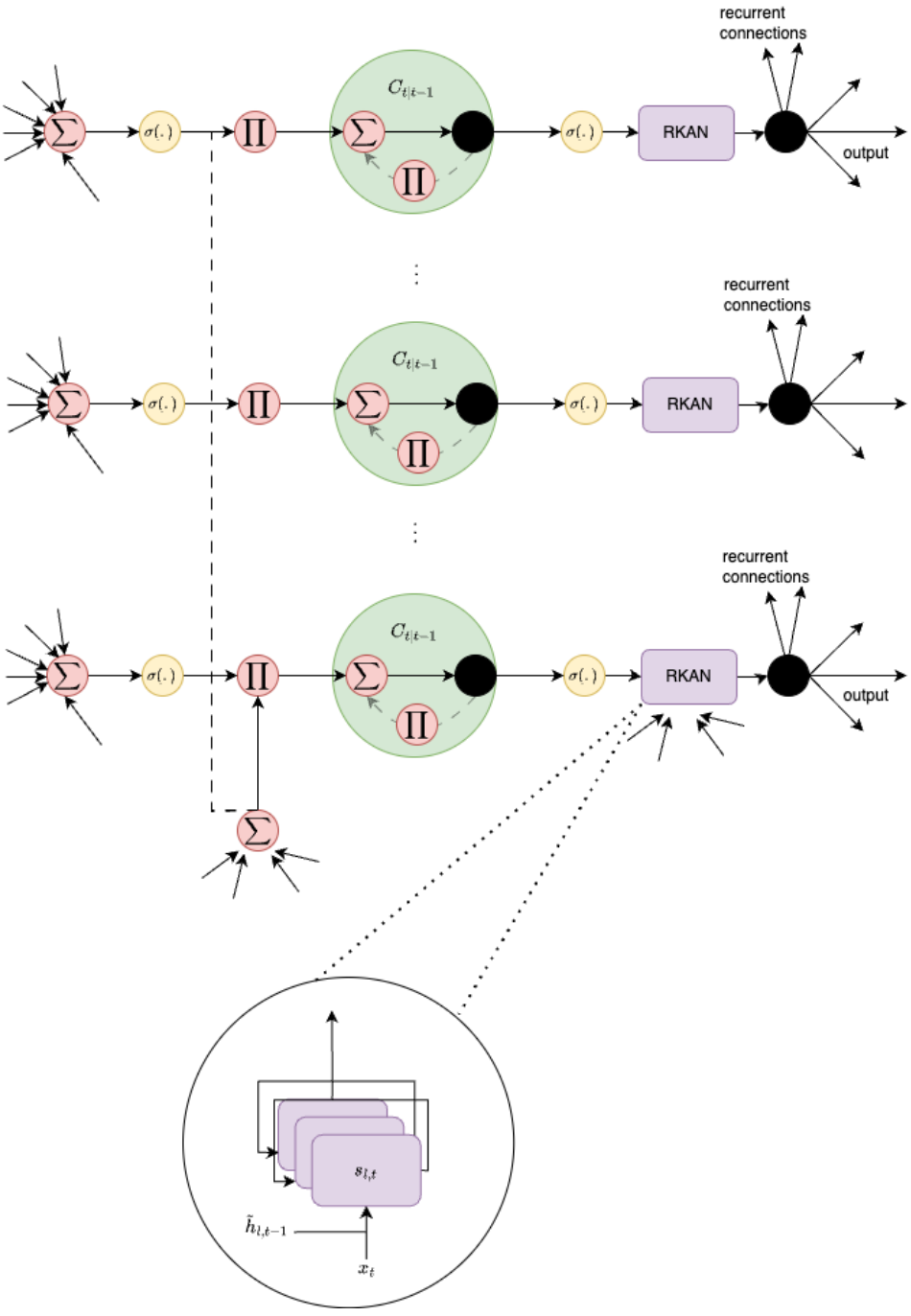


Figure 2.2 – A three layers Temporal Kolmogorov-Arnold Networks (TKAN) Block

For the next step, our aim is to manage memory and we took inspiration from LSTM [HS97b; SM19]. We denote x_t the input vector of dimension d. This unit uses several internal vectors and gates to manage information flow. The forget gate with activation vector f_t given by

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{2.11}$$

decides what information to forget from the previous state. The input gate, with activation vector denoted i_t, with

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{2.12}$$

controls which new information to include. The output gate, with activation vector o_t. Set

$$r_t = \text{Concat}[\phi_1(s_{1,t}), \phi_2(s_{2,t}), ..., \phi_L(s_{L,t})], \tag{2.13}$$

where r_t is a concatenation of the output of multiple KAN Layers, and

$$o_t = \sigma(W_o r_t + b_o), \tag{2.14}$$

where $W_o \in \mathbb{R}^{(\text{KAN}_{out} * L, out)}$, out denoting the output dimension of TKAN. Equation (2.14) determines what information from the current state to output given r_t given from (2.13). $\tilde{h}_t$ is the "sub" memory of the RKAN layers. The hidden state of the TKAN layer, h_t, captures the unit's output, while the cell state c_t is updated as follows:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{2.15}$$

where $\tilde{c}_t = \sigma(W_c x_t + U_c h_{t-1} + b_c)$ represents its internal memory. All these internal states have a dimensionality of h. The output of the final hidden layer, denoted h_t, is given by:

$$h_t = o_t \odot \tanh(c_t). \tag{2.16}$$

In the following section, we will describe the learning task and proceed several tests. We started to test the power of prediction of our model for one step ahead forecasting and in the second time for multi-step ahead [Fan+19; Lim+21].

4 Learning task

In order to assess the relevance of extending our model, we created a simple training task to judge its ability to improve the out-of-sample prediction accuracy over several steps ahead compared with standard layers such as GRU or LSTM. We have chosen to carry out our study not on synthetic data, but rather on real market data, as these seem more relevant to us. Indeed, synthetic market data can be easily biased to match an experiment.

4.1 Task Definition and Dataset

The task is to predict the market notional Bitcoin trades over future periods. This is recognized as a difficult task as the market behavior is difficult to predict. Nonetheless, this should not be impossible as, unlike returns, volumes have internal patterns such as seasonality, autocorrelation, and so on. We used Binance as our only data source, as it is the largest player in the crypto-currency market. Moreover, due to the lack of an overall regulation that would take into account all exchange data, exchanges are subject to a lot of falsified data by small players

who use wash trading to boost their figures.

Our dataset consists of the notional amounts traded each hour on several assets: BTC, ETH, ADA, XMR, EOS, MATIC, TRX, FTM, BNB, XLM, ENJ, CHZ, BUSD, ATOM, LINK, ETC, XRP, BCH and LTC, which are to be used to predict just one of them, BTC. The data period runs from January 1, 2020, to December 31, 2022.

4.2 Preprocessing

Data preparation is a necessary step for most machine learning methods, in order to help gradient descent calculations when data are of different sizes, to obtain stationarity series, etc. This is even more true when using the Kolmogorov-Arnold network, as the underlying B-Spline activation functions exhibit power exponent. Thus, having poorly scaled data would result in over or underflows that would hinder learning. This is also true for market volume data, or notional data. Indeed, they are series with very different scales across assets and between points in time, not to mention non-stationarity issues over a long period.

To obtain data that can be used for training, but also return meaningful losses to optimize, we use a two-stage scaling. The first step is to divide the values in the series by the moving median of the last two weeks. This moving median window is also shifted by the number of steps forward we want to predict, so as not to include foresight. This first pre-treatment aims to make the series more stationary over time. The second pre-processing we apply is a simple MinMaxScaling per asset: even if the minimum of the series is 0, this way of working is simply a matter of dividing by its maximum value. The objective is to scale the data in the $[0, 1]$ interval to avoid an explosive effect during learning due to the power exponent. This pre-processing is, however, adjusted on the training set, the adjustment meaning only the search for the maximum value of each series, which is then used directly on the test set. This means that, on the test set, it is possible to obtain observations that are greater than 1. As long as no optimization is launched, this is not a problem. Finally, we split our dataset into a training set and a test set, with a standard proportion of 80-20. This represents over 21,000 points in the training set and 5,000 in the test set.

4.3 Loss Function for Model Training

Since we have a numerical prediction problem, we have opted to optimize our model using the root mean square error (RMSE) as the loss function, whose formula is simple: RMSE is the square root of

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{X}_{t+1}^{(i)} - X_{t+1}^{(i)}\right)^2,$$

where N represents the number of samples in the dataset, $\hat{X}_{t+1}^{(i)}$ denotes the predicted notional values of Bitcoin at time $t+1$ for the i-th sample, and the quantities $X_{t+1}^{(i)}$ are the corresponding true values. We invoke RMSE first because it is the most widely used and standard loss in machine learning for this type of problem. The second reason is related to the metric we want

to use to display the results, namely the R-squared (R^2). Indeed, (R^2) is interesting as a metric because it not only gives information on the error but also on the error given the variance of the estimated series, which means it's much easier to tell whether the model is performing well or not. This explains why it is a measure widely used by econometricians and practitioners. However, minimizing MSE is exactly the same problem as maximizing the (R^2), as its formula indicates:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(\hat{X}_{t+1}^{(i)} - X_{t+1}^{(i)})^2}{\sum_{i=1}^{N}(X_{t+1}^{(i)} - \bar{X}_{t+1})^2},$$

4.4 Benchmarks

4.4.1 Model Architectures

In order to compare with comparable things, we tested our TKAN layers against two of the most widely used RNNs for multi-step predictions, namely gated recurrent units (GRU) and long-term short-term memory (LSTM). We are not comparing our work to complete model architectures such as a temporal fusion transformer, as what we are proposing is more a layer than a complete model architecture.

To fairly compare the three models, we have opted for a very simple configuration. We create three models to be compared in the same way:

1. An initial recurrent layer of 100 units that returns complete sequences,
2. An intermediate recurrent layer of 100 units, which returns only the last hidden state,
3. A final dense layer with linear activation, with as many units as there are timesteps to predict ahead

For the TKAN model, we used 5 B-spline activations of order 0 to 4 as sublayer activations, while we used the standard activation function for GRU and LSTM. Finally, we also compared the three models to the most naive benchmark, which consists of using the last value as a predictor of the future, the value being repeated when using several predictions in advance.

4.4.2 Note on training details

Metrics are calculated directly on scaled data and not on unscaled data. There are two reasons for this: firstly, MinMax scaling has no impact on the metric since the minimum is 0 and the data interval is $[0, 1]$; rescaling would not have changed the R-squared. Rescaling from the median split would have caused the series mean to be unstable over time. This would have resulted in the error magnitude drifting for certain parts of the series, making the metric unreliable or meaningless. In terms of optimizing model details, we used the Adam optimizer, one of the most popular choices, used 20% of our training set as a validation set and included two training recalls. The first is an early learning stopper, which interrupts training after 6 consecutive periods without improvement on the validation set, and restores the weights associated with the best loss obtained on the validation set. The second is a plateau learning rate reduction, which halves the learning rate after three consecutive periods showing no improvement on the validation set.

4.5 Results

To evaluate the model's performance, taking into account the risk of poor adaptation that may occur, we repeat the experiment 5 times for each, and display below the mean and standard deviation of the training results obtained from these 5 experiments.

4.5.1 Performance Metrics Summary

Table 2.1 – Average (R^2) obtained over 5 run

Time	TKAN:5 B-Spline	GRU:default	LSTM:default	Last Value
1	0.33736	0.365136	0.355532	0.292171
3	0.21227	0.200674	0.061220	-0.062813
6	0.13784	0.082504	-0.225838	-0.331346
9	0.09803	0.087164	-0.290584	-0.457718
12	0.10401	0.017864	-0.473220	-0.518252
15	0.09512	0.033423	-0.404432	-0.555633

Table 2.2 – Standard Deviation of the (R^2) obtained over 5 run

Time	TKAN:5 B-Spline	GRU:default	LSTM:default
1	0.00704	0.008336	0.011163
3	0.00446	0.004848	0.080200
6	0.01249	0.023637	0.062710
9	0.02430	0.014833	0.052729
12	0.00132	0.086386	0.085746
15	0.00701	0.024078	0.092729

The results show a very logical decrease in terms of R2 with the number of forward steps ahead, which is quite normal since we have less information for the forward steps. However, the results clearly show two things: the first is that while all models have a relatively small difference in performance over a short horizon such as 1 to 3 steps ahead, the relative performances change much more with a longer time horizon. The LSTM model even becomes useless and counter-productive with 6 periods, while TKAN and GRU always achieve a higher average R-squared value. However, TKAN stands out with longer time steps, with an R-squared value at least 25% higher than that of GRU. Another very interesting point is model stability, i.e. the ability to calibrate well-functioning weights from samples without too much variation from one experiment to the next, and here again, TKAN shows much better stability than all the other models.

4.5.2 Training Dynamics and Model Stability

To better understand the differences in performance between the models, we visualized training and validation losses over several training sessions for each model. These graphs offer a dynamic view of each model's learning process and ability to generalize beyond the training data.

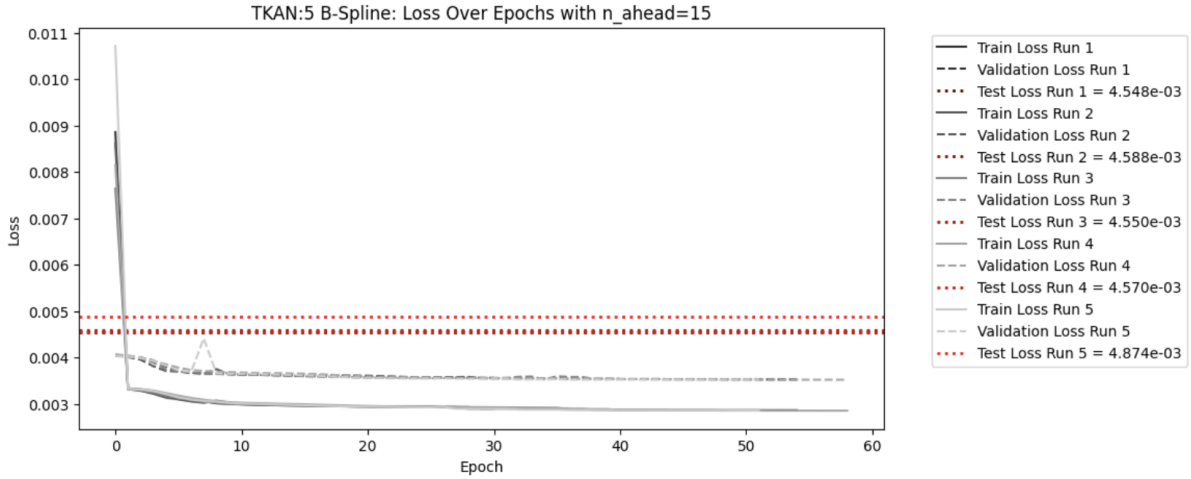


Figure 2.3 – TKAN training and validation loss over epochs

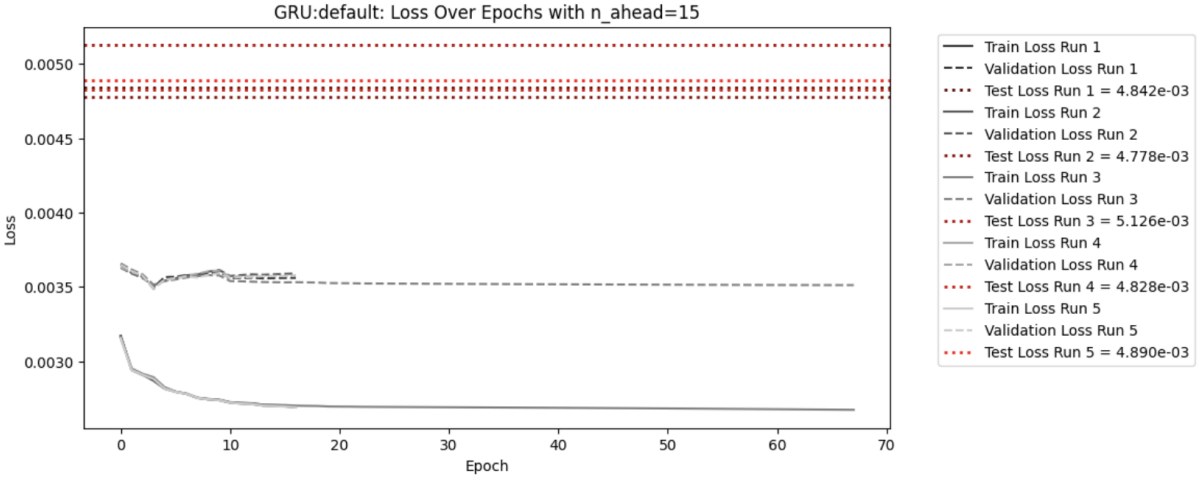


Figure 2.4 – GRU training and validation loss over epochs

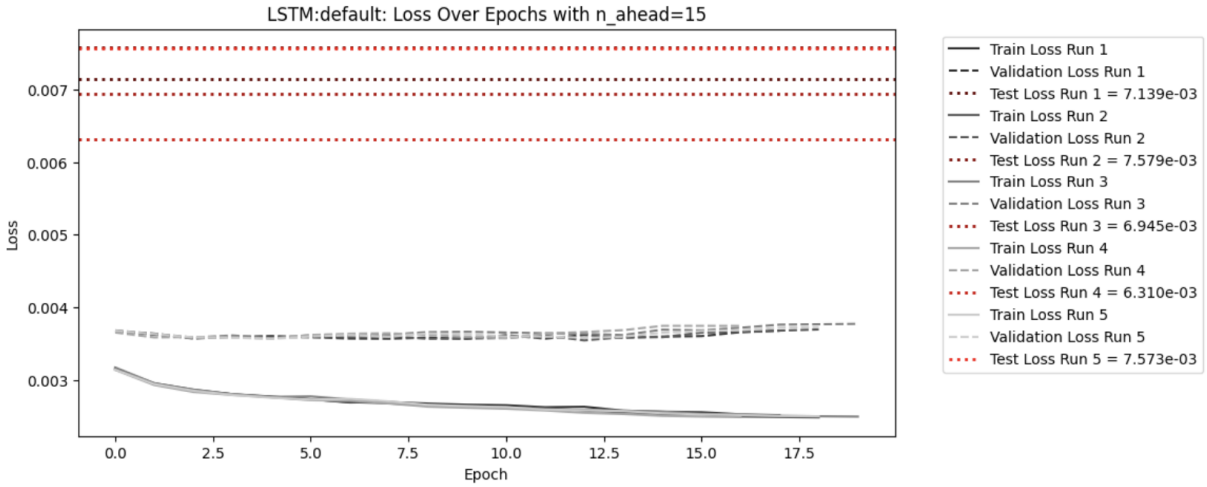


Figure 2.5 – LSTM training and validation loss over epochs

The visual representations clearly corroborate the statistical results presented above. The GRU and LSTM models show a significant divergence between their learning loss and validation trajectories, particularly as the number of epochs increases. This divergence suggests a potential overfitting where the model learns idiosyncrasies from the training data rather than generalizing them. This stability in the TKAN model's learning process, evident in the closer alignment of its learning and validation loss curves, implies a consistent learning model that effectively captures the underlying patterns in the data without overfitting.

5 Conclusion

In this chapter, we proposed an adaptation of the Kolmogorov-Arnold Network architecture for time series that incorporates both recurring and gating mechanisms. The architecture, while not so complicated, enables improving multiple-step performances and stability compared to traditional methods and seems to be promising. The temporal Kolmogorov-Arnold networks (TKANs) combine the best features of recurrent neural networks (RNNs) and Kolmogorov-Arnold Networks (KANs). This new architecture tackles the usual problems of RNNs (long-term dependency). TKANs embed Recurrent Kolmogorov-Arnold Networks (RKAN). These layers help the system to memorize and use new and old information efficiently. Compared to more traditional models such as LSTM and GRU, TKAN particularly stands out when it comes to making longer-term predictions, showing that it is capable of handling different situations and longer periods of time. Our experiments show that it is usable and more stable than GRU and LSTM on real historical market data. While not specifically interesting for short-term predictions, it especially demonstrates an ability to largely outperform other models when it comes to multi-step predictions.This also confirms that the idea developed in the original KAN paper works well on real use cases and is totally relevant for time series analysis. This paper opens interesting new ways to improve our capacities to calibrate accurate time-series models over multiple steps, which is one of the hardest tasks in temporal analysis.

Chapter 3

A Temporal Kolmogorov-Arnold Transformer for Time Series Forecasting

This part is a joint work with Rémi Genet (Univ. Paris Dauphine).

Abstract

Capturing complex temporal patterns and relationships within multivariate data streams is a difficult task. We propose the Temporal Kolmogorov-Arnold Transformer (TKAT), a novel attention-based architecture designed to address this task using Temporal Kolmogorov-Arnold Networks (TKANs). Inspired by the Temporal Fusion Transformer (TFT), TKAT emerges as a powerful encoder-decoder model tailored to handle tasks in which the observed part of the features is more important than the a priori known part. This new architecture combined the theoretical foundation of the Kolmogorov-Arnold representation with the power of transformers. TKAT aims to simplify the complex dependencies inherent in time series, making them more "interpretable". The use of transformer architecture in this framework allows us to capture long-range dependencies through self-attention mechanisms.

Contents

1 Introduction

Multivariate time series forecasting is an area of strong interest for researchers [TSB10; TT89; Wei18; MSW06] and has become an important area of research for many industries. The volume of available data is increasing. Thus, in some way, it is natural to propose more and more advanced frameworks for prediction purposes. In contrast to univariate time series, in which a single variable is analyzed over time, multivariate time series involve several interdependent variables. These connections make the learning task even more complex. Multivariate time series data require sophisticated models that can capture dynamic interactions and temporal dependencies across multiple dimensions and time horizons. Researchers have explored various methods, including vector autoregressive models [ZW06; Lüt13], state-space models [Ham94; Kim94], and neural networks [TDF91], to tackle the challenges associated with multivariate time series forecasting, i.e., to identify complex dependencies among time series [BMD18; ML19]. Deep learning approaches such as Long Short-Term Memory (LSTM) networks and attention mechanisms have shown promising results due to their ability to handle non-linearities and long-range dependencies in dependent data [HS97b; Vas+17]. These advanced techniques are crucial for improving forecasting accuracy and making informed decisions based on the predicted outcomes.

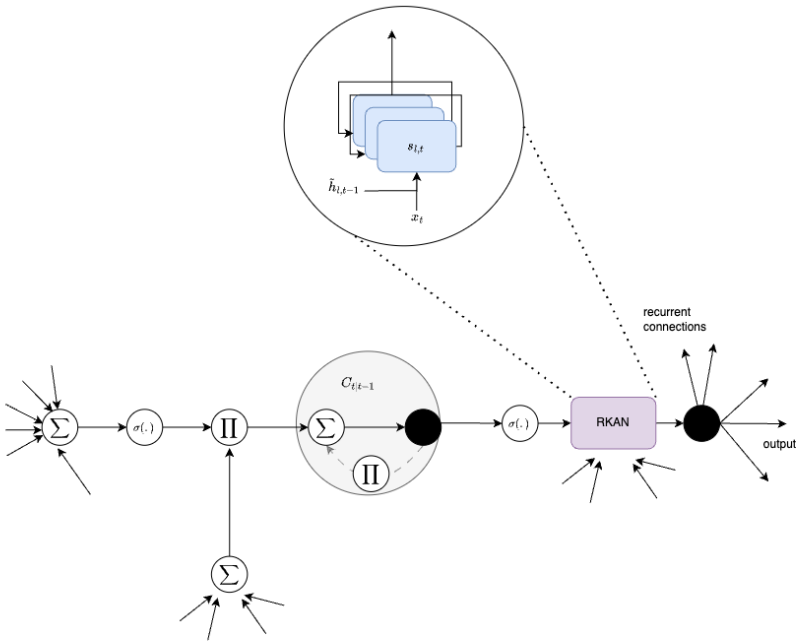


Figure 3.1 – Temporal Kolmogorov-Arnold Networks (TKAN)

To improve forecasting accuracy, some researchers have suggested employing attention mech-

anisms [SSL19] or convolutional methods for multivariate time series prediction [Wan+19a]. More recently, Liu et al. [Liu+24] released the Kolmogorov-Arnold Networks (KANs) a promising architecture, introduced as an alternative for MLPs. To dive into the architecture of the model, we redirect the reader to [Liu+24]. KANs outperform conventional MLPs in real-world forecasting tasks [Vac+24]. Even more recently, we introduced an extension, incorporating time dependency as well as memory management [GI24b]. Another extension using wavelets [BC24] has been proposed in the meantime. TKANs aim to develop a framework with two key functionalities: handling sequential data and managing memory. To do so, [GI24b] enrich the initial model by adding a recurring layer of KANs to introduce RKANs. Temporal Kolmogorov-Arnold Networks (TKANs) are an upgraded version of LSTM [HS97b] using Kolmogorov-Arnold Networks (KANs). They rely on RKANs for the management of short-term memory as well as a cell state; for further details, we refer the reader to [GI24b]. While TKAN layers stacked in TKANs are capable of detecting temporal dependencies in time series data, it seems natural to propose an alternative to the Temporal Fusion Transformer [Lim+21] using TKAN layers instead of LSTM layers. Our codes are available at TKAT repository and can be installed using the following command: `pip install tkat`. The data are accessible if a reader wishes to reproduce the experiments inside the github provided above.

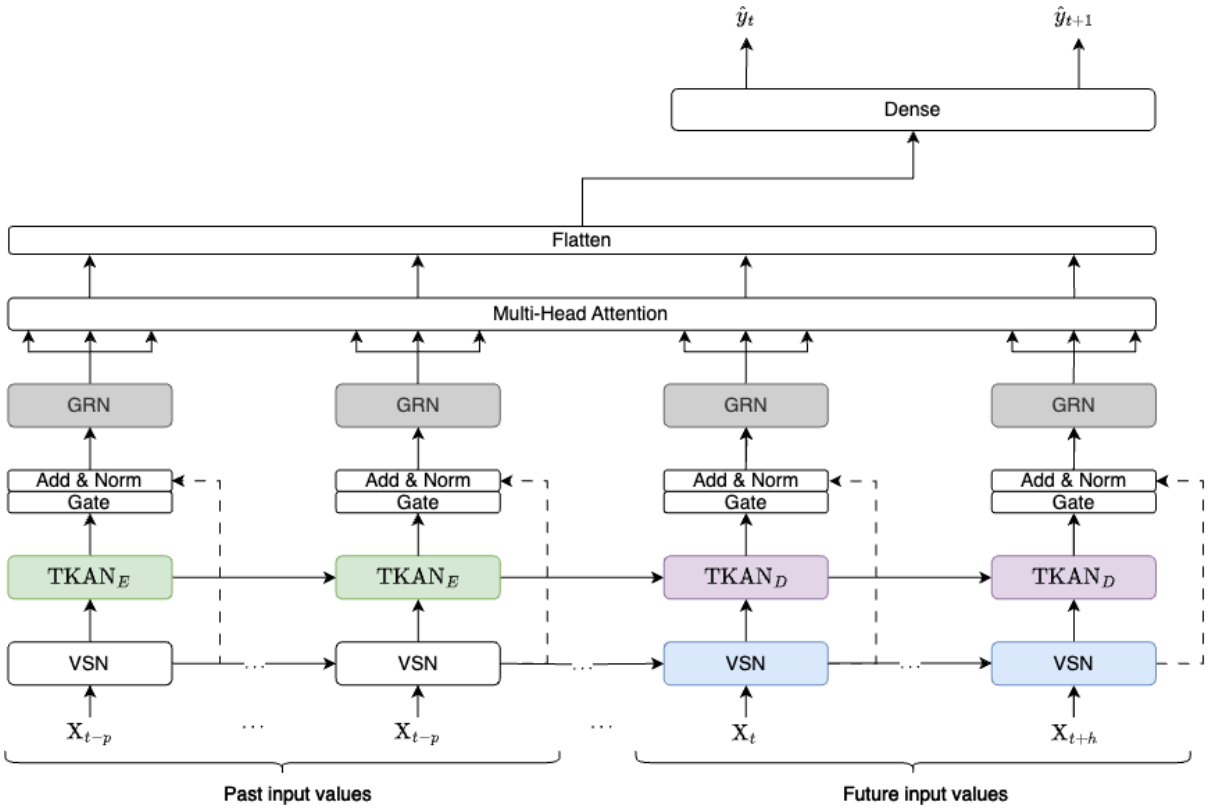


Figure 3.2 – Temporal Kolmogorov-Arnold Transformer (TKAT)

As said previously, time series forecasting is a complex task. Managing dependencies within a time series has been considered the biggest challenge for many modelers. RNNs appeared

to be the most appropriate model to tackle the dependency problem. In 2017, Transformers [Vas+17] have been introduced to tackle such issues in natural language processing (NLP), translation [Meh+20; Raf+20; SLL19; Fan+21], language modeling [Sho+19; Dai+01; Roy+21; Rae+19] and named entity recognition [Yan+19; Ark+19; Yu+20]. A Transformer-based approach has been introduced [Lim+21; Wu+20a] to forecast time series. Compared to other sequence-aligned deep learning methods, this architecture is leveraging self-attention mechanisms to model sequences and make possible to learn complex dependencies between various lagged realizations of time series. The Transformer architecture has increased the quality of prediction. Nonetheless, the latter architecture suffers from severe issues that prevent it from being directly applicable to Long-Term Sequence Forecasting (LSTF), including quadratic time complexity, high memory usage, and inherent limitation of the encoder-decoder architecture. To overcome such problems [Zho+21] introduced an efficient transformer-based model for LSTF. To address the issue of memory usage, [Li+19] proposed a LogSparse Transformer, which has a memory cost of just $O(L(logL)^2)$. This model enhances the forecasting accuracy for time series data with fine granularity and significant long-term dependencies, while operating under a constrained memory budget. Another transformer-based model is given by the Adversarial Sparse Transformer(AST)[Wu+20b], a novel architecture for time series forecasting. They propose Sparse Transformer to improve the ability to pay more attention to some relevant steps in time series. By adversarial learning, they improve the fit at the sequence level. To address these challenges, we propose the Temporal Kolmogorov-Arnold Transformer (TKAT), a novel architecture inspired by the temporal fusion transformer [Lim+21] which integrates the theoretical foundation of the Kolmogorov-Arnold representation [Kol61] with the power of transformers. TKAT aims to simplify the complex dependencies inherent in time series, making them more "interpretable". The use of the transformer's architecture in this framework allows us to capture long-range dependencies through self-attention mechanisms.

In this paper, we will propose a novel approach for temporal transformers. We will use the Multi-Head Attention mechanism [Vas+17], but with updates on encoder-decoder phases and temporal processing. Our approach aims to integrate the effective architecture of the TFT model with the TKAN one, the latter framework having demonstrated good performance in n-step ahead forecasting.

2 Model Architecture

We propose a new transformer architecture using the layers of the Temporal Kolmogorov-Arnold Network [GI24b] inspired by the Temporal Fusion Transformer (TFT)[Lim+21]. The general idea is to use TKAN instead of LSTM for the encoding and decoding layers, but also to propose an architecture adapted to problems where "known inputs" or deterministic inputs are not very common, unlike observed variables, which are very widespread in financial data. This is a major difference from the standard task for which the TFT was designed. We have observed here that, for series with few known inputs, our architecture offers much better performance.

2.1 TKAN

In this subsection, we recall the TKAN definition, detailed in Chapter 2.

Recurrent Kernel is key for RKAN layers learnt from sequences where context and order of data matter. We have designed TKAN to leverage the power of Kolmogorov-Arnold Network while offering memory management to handle time dependency. To introduce time dependency, each transformation function $\phi_{l,j,i}$ has to be time dependent. Let us denote the sub layers memory state $\tilde{h}_{l,t}$, initialized with zeros and of shape (KAN_{out}). The inputs that are fed to each sub KAN layers in the RKAN are created as follows:

$$s_{l,t} = W_{l,\tilde{x}} x_t + W_{l,\tilde{h}} \tilde{h}_{l,t-1}, \tag{3.1}$$

where $W_{l,\tilde{x}}$ is the weight of the l-th layer applied to x_t, which is the input at time t. $W_{l,\tilde{h}}$ contain the weights of the l-th layer applied to its previous substate. Let us first denote KAN_{in} and KAN_{out} the input dimensions of RKAN Layer input and ouputs, respectively. We have here $W_{l,\tilde{x}} \in \mathbb{R}^{(d,\text{KAN}_{in})}$ and $W_{l,\tilde{h}}^{(\text{KAN}_{out},KAN_{in})}$, which leads to $s_{l,t}^{\text{KAN}_{in}}$.

$$\tilde{o}_t = \phi_l(s_{l,t}), \tag{3.2}$$

where ϕ_l is a KAN layer. The "memory" step $\tilde{h}_{l,t}$ is defined as a combination of past hidden states, such as

$$\tilde{h}_{l,t} = W_{hh} \tilde{h}_{l,t-1} + W_{hz} \tilde{o}_t, \tag{3.3}$$

where W is a vector of weights. W weights the importance of past values relatively to the most recent inputs. For the next step, to maintain memory, we took inspiration from the LSTM [HS97b; SM19] technique. We denote x_t the input vector of dimension d. This unit uses several internal vectors and gates to manage information flow. The forget gate, with activation vector f_t, i.e.,

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{3.4}$$

decides what information is forgotten from the previous state. The input gate, with activation vector denoted i_t,

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{3.5}$$

controls new information to be included. The output gate, with activation vector o_t,

$$r_t = \text{Concat}[\phi_1(s_{1,t}), \phi_2(s_{2,t}), \ldots, \phi_L(s_{L,t})]. \tag{3.6}$$

r_t concatenates the outputs of multiple KAN Layers and will feed an activation function to propagate a part of the information. Thus, we define

$$o_t = \sigma(W_o r_t + b_o), \tag{3.7}$$

where $W_o \in \mathbb{R}^{(\text{KAN}_out*L,out)}$, with *out* the ouput dimension of TKAN. Equation (3.7) determines

the retained information from the current state to the output, given r_t coming from (3.6). $\tilde{h}_t$ is the "sub" memory of the RKAN layers. The hidden state h_t of the TKAN layer captures the unit's output, while the cell state c_t is updated as

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{3.8}$$

where $\tilde{c}_t = \sigma(W_c x_t + U_c h_{t-1} + b_c)$ represents its internal memory. All these internal states have a dimensionality of h. The ouput of the final hidden layer, denoted h_t is given by

$$h_t = o_t \odot \tanh(c_t). \tag{3.9}$$

2.2 Encoder-Decoder

Encoding and Decoding tasks allow sequence-to-sequence mapping (i.e. machine translation, text mining, etc.). Picking the use case of machine translation, the global idea of the task is to convert an input sequence into a target sequence. In TKAT, we use layers of TKAN detailed in Section 3 for both encoding and decoding tasks.

2.2.1 Encoder

The encoder is composed of single or multiple layers of TKAN_E. The recurring mechanism within the layer allows monitoring the information and keeping specific parts of the sequences, and so to capture long-range dependencies. The encoder will be fed by the past inputs (observed/known inputs) filtered by the variable selection networks (VSN) detailed in a later section.

2.2.2 Decoder

Like the encoder, the decoder consists of several layers of TKAN_D. Each layer allows processing information while maintaining the memory and time dependency. However, the decoder will take as an initial state the encoder's final cell state.

2.3 Gated Residual Networks

The relationships between dependent random vectors is a key issue. Gated residual networks offer an efficient and flexible way of modelling complex relationships in time series. They allow to control the flow of information and facilitate the learning tasks. They are particularly useful in areas where non-linear interactions and long-term dependencies are crucial. In our model we use the Gated Residual Networks proposed by [Lim+21], and we kept the same architecture. However, we remove the need for the context.

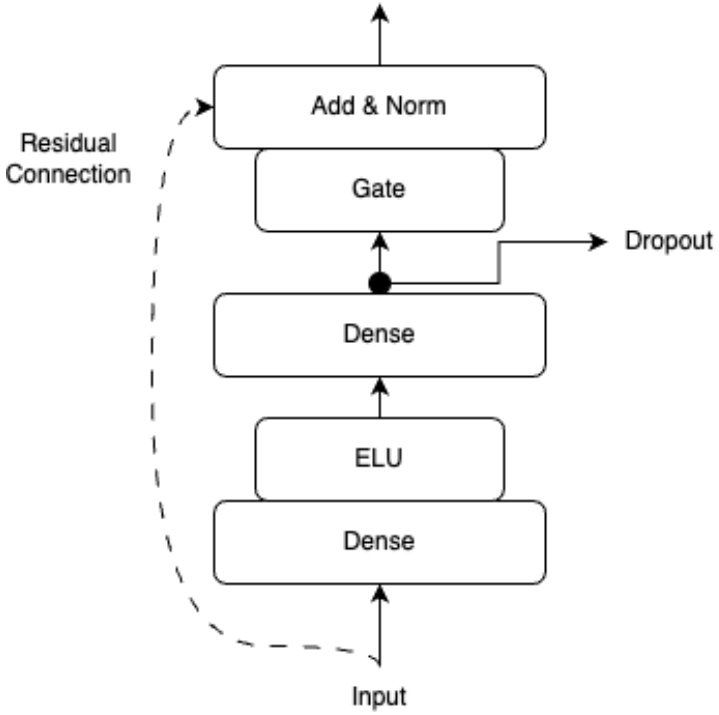


Figure 3.3 – Gated Residual Networks (GRN) [Lim+21]

$$\mathrm{GRN}_\omega(x) = \mathrm{LayerNorm}\left(x + \mathrm{GLU}_\omega(\eta_1)\right), \tag{3.10}$$

$$\eta_1 = W_{1,\omega}\ \eta_2 + b_{1,\omega}, \tag{3.11}$$

$$\eta_2 = \mathrm{ELU}\left(W_{2,\omega}\ x + b_{2,\omega}\right). \tag{3.12}$$

In this context, ELU designates the Exponential Linear Unit activation function [CUH15], while $\eta_1 \in \mathbb{R}^{d_{model}}$ and $\eta_2 \in \mathbb{R}^{d_{model}}$ represent intermediate layers. The standard layer normalization LayerNorm is that described in [BKH16], and ω is an index used to indicate weight sharing. When the expression $W_{2,\omega}x + b_{2,\omega}$ is largely positive, ELU activation works as an identity function. On the other hand, when this expression is largely negative, ELU activation produces a constant output, thus behaving like a linear layer. We use Gated Linear Units (GLUs) [Dau+17]. They provide the flexibility to suppress any parts of the architecture that are not required for a given dataset. Letting $\gamma \in \mathbb{R}^{d_{model}}$ be the input, the GLU then takes the form

$$\mathrm{GLU}_\omega(\gamma) = \sigma(W_{4,\omega}\ \gamma + b_{4,\omega}) \odot (W_{5,\omega}\ \gamma + b_{5,\omega}), \tag{3.13}$$

where $\sigma(.)$ is the sigmoid activation function; $W(.) \in \mathbb{R}^{d_{model} \times d_{model}}$, $b(.) \in \mathbb{R}^{d_{model}}$ are weights and intercepts, respectively; $\odot$ denotes the element-wise Hadamard product, and d_{model} is the hidden state size (common across TFT). GLU allows TFT to control the extent to which the GRN contributes to the original input x – potentially skipping over the layer entirely if necessary, as the GLU outputs could be all close to 0 in order to suppress the nonlinear contribution of

input variable to forecasting.

2.4 Variable Selection Networks

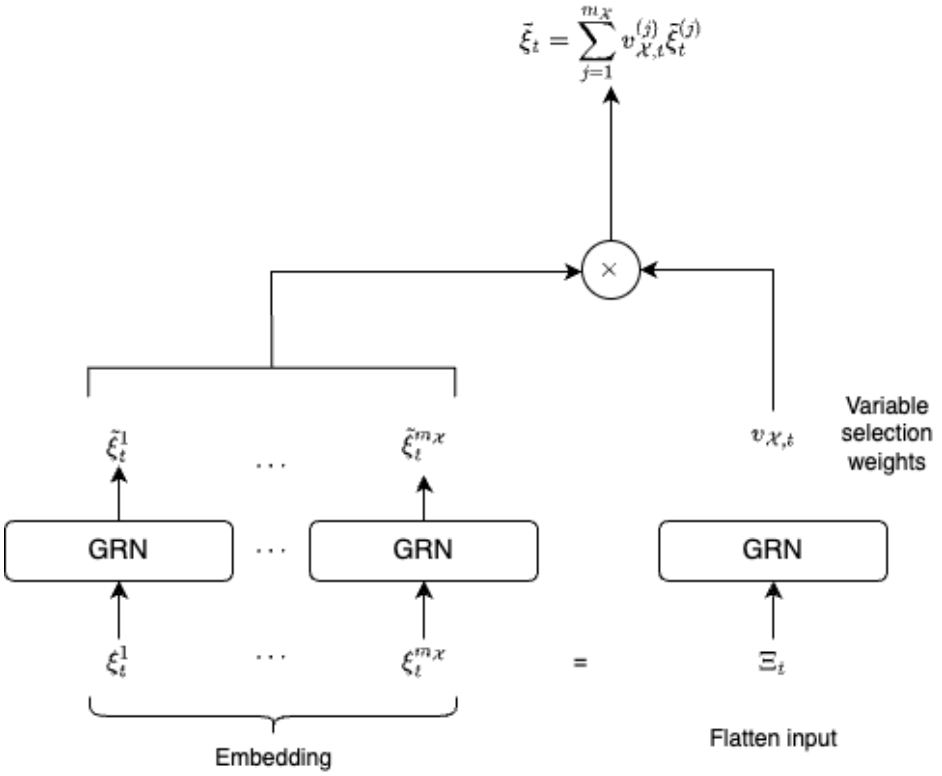


Figure 3.4 – Variable Selection Networks (VSN)

The Variable Selection Networks (VSN) will help model performance via the use of learning capacity only for the most salient covariates. Figure 3.4 described the different steps in the VSN. We define $\xi_t^{(j)} \in \mathbb{R}^{d_{model}}$ the transformed input of the j-th variable at time t, with $\Xi_t = \left[\xi_t^{(1)^T}, \ldots, \xi_t^{(m_\chi)^T}\right]^T$, the flattened vector of all past inputs at time t. Variable selection weights, denoted $v_{\mathcal{X},t}$, are generated by feeding both Ξ_t and an external context vector c_s through a GRN and followed by a Softmax layer:

$$v_{\chi_t} = \text{Softmax}\left(\text{GRN}_{v_\chi}(\Xi_t)\right), \tag{3.14}$$

where $v_{\chi_t} \in \mathbb{R}^{m_\chi}$ is a vector of variable selection weights. We remove the need for a static covariate encoder as we are not embedding static covariates within our model. For each time step, a non-linear processing is employed by feeding each $\xi_t^{(j)}$ through its own GRN:

$$\tilde{\xi}_t^{(j)} = \text{GRN}_{\tilde{\xi}(j)}\left(\xi_t^{(j)}\right), \tag{3.15}$$

where $\tilde{\xi}_t^{(j)}$ is the processed feature vector for variable j. We note that each variable has its own $\text{GRN}_{\xi(j)}$, with *weights shared across all time steps* t. Processed features are then weighted by

their variable selection weights and combined. This yields

$$\tilde{\xi}_t = \sum_{j=1}^{m_\chi} v_{\chi_t}^{(j)} \tilde{\xi}_t^{(j)}, \tag{3.16}$$

where $v_{\chi_t}^{(j)}$ is the j-th element of vector v_{χ_t}.

2.5 Temporal Decoder

The temporal decoder in the Temporal Fusion Transformer generates accurate and interpretable time series forecasts. This is mainly due to using attention mechanisms, positional encodings, static context, and recurrent neural units to capture short-term and long-term dependencies. The TKAT will also embed a temporal decoder to make accurate forecasts. However, the structure of the temporal decoder will be slightly different.

2.5.1 Self-Attention Layer

The TKAT uses a self-attention mechanism to capture long-term relationships between different time steps, modified from multi-head attention in transformer-based architectures [Li+19].

$$\text{Attention}(Q, K, V) = A(Q, K)V, \tag{3.17}$$

where $A(.)$ is a normalization function. A common choice is the scaled dot-product attention [Vas+17]:

$$A(Q, K) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_{attn}}}\right). \tag{3.18}$$

To enhance the learning capacity of the standard attention mechanism, [Vas+17] proposed multi-head attention, which employs different heads for different representation subspaces:

$$\text{MultiHead}(Q, K, V) = [H_1, \ldots, H_{m_H}]W_H, \tag{3.19}$$

$$H_h = \text{Attention}(QW_Q^{(h)}, KW_K^{(h)}, VW_V^{(h)}), \tag{3.20}$$

where $W_K^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}$, $W_Q^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}$, and $W_V^{(h)} \in \mathbb{R}^{d_{model} \times d_V}$ are head-specific weights for keys, queries, and values, respectively. The matrix $W_H \in \mathbb{R}^{(m_H \cdot d_V) \times d_{model}}$ linearly combines outputs concatenated from all heads H_h. Given that different values are used in each head, attention weights alone would not indicate a particular feature's importance.

2.5.2 Fully Aware Layer

In time series analysis, past values play a very important role in establishing reliable predictions. Historical data can capture trends, seasonality, and the underlying patterns that will impact the evolution of a variable through time. Past observations also allow us to establish

relationships and interdependencies. This led us to modify this layer by proposing to flatten the output of the Temporal self-attention layer. This modification allows the network to be fully connected to past values and not just to the unknown inputs fed by the encoder's final cell state. This architecture enables us to maintain a more persistent memory. Given the output of the temporal self-attention layer $[\tilde{H}_1, \tilde{H}_2, \ldots, \tilde{H}_{m_H}]$, where each $\tilde{H}_i$ is a vector in $\mathbb{R}^{m_i}$ and $i = 1, \ldots, H$, the flattened vector $\tilde{H}_{\text{flat}}$ in $\mathbb{R}^{m_1+m_2+\cdots+m_H}$ can be written as

$$\tilde{H}_{\text{flat}} = \begin{bmatrix} \tilde{H}_1 \\ \tilde{H}_2 \\ \vdots \\ \tilde{H}_{m_H} \end{bmatrix}. \tag{3.21}$$

2.6 Ouputs

The output of the TKAT is obtained from the flattened vector of MultiHeadAttention(.):

$$\hat{y}_{t:t+\tau} = W_{\hat{y}} \tilde{H}_{\text{flat}} + b_{\hat{y}}, \tag{3.22}$$

where $W_{\hat{y}} \in \mathbf{R}^{\tau}$. The proposed architecture provides a sound framework to forecast multivariate time series data n-step ahead. The fully connected linear layer output dimension corresponds to the number of steps $k = [1, \ldots, \tau]$ to be forecast. The output vector $\hat{y}_{t:t+\tau} = [\hat{y}_{t+1}, \ldots, \hat{y}_{t+\tau}]$ contains all the forecasting values for each timestep. Thus, each predicted value is a weighted sum of all the attention results, with different weights for each step.

3 Learning Task

To test the model's ability to offer better predictions with our approach, we used the same tasks as in the article [GI24b], for an easy comparison of the results.

3.1 Task definition and dataset

The task consists of predicting the notional amount traded on the market over several steps ahead. We chose this task because it is far from easy, as market data is notorious for its lack of predictability, even though volume is a process that has predictable components such as seasonality over the day and week, as well as strong auto-correlations.

The tasks focus solely on the Binance exchange, so our dataset only contains data from this exchange, which has been the most important market for many years. We have also only used USDT markets, as this is the most widely used stablecoin, and all notionals are therefore intended in USDT.

Our dataset consists of the notional amounts traded each hour on several assets: BTC, ETH, ADA, XMR, EOS, MATIC, TRX, FTM, BNB, XLM, ENJ, CHZ, BUSD, ATOM, LINK, ETC, XRP, BCH and LTC, which are to be used to predict just one of them, BTC. The data period runs over 3 years from January 1, 2020 to December 31, 2022. However, all these values are

observed values, i.e. they are not known in the future, whereas, as can be seen, the model architecture requires known inputs in the futures. Thus, for the transformer, we have added 2 sets of data containing the time of day and the day of the week, which are known values and therefore present on both past and future inputs.

3.2 Data Preparation

Data preparation is a necessary step for most machine learning methods, to facilitate gradient descent when data are of different sizes, to ensure series stationarity, etc. This is very important in our dataset for two reasons: first, the traded assets are observed at very different scales; second, the notionals of an asset are not at all stationary over time.

In order to obtain data that can be used for training, but also to return a meaningful loss to be optimized, we use a two-step scaling.The first step is to divide the values in the series by the moving median of the last two weeks. This moving median window is also shifted by the number of steps forward we want to predict, so as not to include foresight. This first pre-treatment aims to make the series more stationary over time. The second pre-processing we applied is a simple MinMaxScaling per asset. Even if here the minimum of the series is 0, it is simply a matter of dividing by the maximum value. The aim is to scale the data in the $[0, 1]$ interval to avoid an explosive effect during learning due to the power exponent. This pre-processing is, however, adjusted on the training set, the adjustment meaning only the search for the maximum value of each series, which is then used directly on the test set. This means that it is possible to have data greater than 1 on the test set. As no optimization is used, this is not a problem. Finally, we split our data set into a training set and a test set, with a standard proportion of 80-20. This represents over 21,000 points in the training set and 5,000 in the test set.

3.3 Loss Function for Model Training

Since we have a numerical prediction problem, we have opted to optimize our model using the root mean square error (RMSE) as the loss function. RMSE is the square root of

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{X}_{t+1}^{(i)} - X_{t+1}^{(i)}\right)^2,$$

where N represents the number of samples in the dataset, $\hat{X}_{t+1}^{(i)}$ denotes the predicted notional values of Bitcoin at time $t+1$ for the i-th sample, and $X_{t+1}^{(i)}$ are the corresponding true values.

The first reason is that this is the most widely used and standard method in machine learning for this type of problem. The second reason is the metric we want to use to display the results, namely the R-squared R^2. The R^2 is interesting as a metric because it not only gives information on the error but also on the error given the variance of the estimated series, which means it's much easier to tell whether the model is performing well or not. It is also a measure widely used by econometricians and practitioners for this very reason. However, minimizing the MSE

is exactly the same as maximizing the R^2, as its formula indicates:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(\hat{X}_{t+1}^{(i)} - X_{t+1}^{(i)})^2}{\sum_{i=1}^{N}(X_{t+1}^{(i)} - \bar{X}_{t+1})^2}.$$

As we can see, the upper terms of the quotient are equivalent to the sum of the squared errors, the other two components being fixed, the optimization of the mean squared error is totally similar to the optimization of R^2.

3.4 Benchmarks

3.4.1 Model Architectures

To assess the merits of our model, we have tested it on various benchmarks. Firstly, we are reusing those of [GI24b], as this is an important baseline, and here we are also comparing the difference with the variant that includes the TKAN layer versus a variant that does not include it. Finally, we have added a few points of comparison to justify why we removed the top part of a standard temporal fusion-transformation architecture with result tables if the top parts were retained.

In order to analyze the performance of our model, here is the model tested:

1. TKAT: refers to the paper model implementation, which uses TKAN layers and flattening directly after multi-head attention. We use a number of hidden units of 100 in each layer, with the exception of the embedding layer, which has only one unit per feature. The number of heads used is 4.
2. TKATN: follows the same implementation, but replaces the TKAN layers with standard LSTM layers to see whether the TKAN layers really improve the model or not.
3. TKAN, GRU and LSTM: Simple model described in [GI24b] consisting of two 100-unit recurrent layers finalized by a dense linear layer.
4. TKAT-A, TKATN-A: Variant that adds a Gate, Add and Normalization layer after the multihead with feedforward of the multihead's attention input. This shows why we have removed this part of our implementation.
5. TKAT-B, TKATN-B: Similar to version A above, with a GRN after the Gate, Add and Normalization, and a final Gate, Add, Normalization with feedforward of the pre-multihead GRN inputs. This architecture is almost entirely similar to the TFT architecture, except that the entire sequence is used after the multihead attention layer, whereas it is only used for the future part in the TFT architecture.

It is important to note that we did not use dropout in any of the above models.

3.4.2 Note on training details

Such in [GI24b], Metrics are calculated directly on scaled data and not on unscaled data. There are two reasons for this: firstly, MinMax scaling has no impact on the metric since the

minimum is 0 and the data interval is [0,1]; rescaling would simply have involved multiplying all the points, which would not have changed the R-squared. Nor do we rescale from the median split, as otherwise the series mean would not have been stable over time, leading to a drift in error magnitude on certain parts of the series, rendering the metric insignificant.In terms of optimizing model details, we used the Adam optimizer, one of the most popular choices, used 20% of our training set as a validation set and included two training recalls. The first is an early learning stopper, which interrupts training after 6 consecutive periods without improvement on the validation set and restores the weights associated with the best loss obtained on the validation set. The second is a plateau learning rate reduction, which halves the learning rate after 3 consecutive periods showing no improvement on the validation set.

4 Results

The results will be illustrated in the latter section, all the methodology, data, and codes are available to reproduce the experience. In order to evaluate the model's performance, taking into account the randomness induced by the weight initializer, we repeat the experiment 5 times for each and display below the mean and standard deviation of the train results obtained from these 5 experiments.

4.1 Performance metrics TKAT summary vs. benchmarks

Table 3.1 – R^2 Average: TKAT vs Benchmark

Time	TKAT	TKAT non-KAN	TKAN	GRU	LSTM
1	0.30519	0.29834	0.33736	0.36513	0.35553
3	0.21801	0.22146	0.21227	0.20067	0.06122
6	0.17955	0.17584	0.13784	0.08250	-0.22583
9	0.16476	0.15378	0.09803	0.08716	-0.29058
12	0.14908	0.15179	0.10401	0.01786	-0.47322
15	0.14504	0.12658	0.09512	0.03342	-0.40443

Table 3.2 – R^2 Standard Deviation: TKAT vs Benchmark

Time	TKAT	TKAT non-KAN	TKAN	GRU	LSTM
1	0.01886	0.01610	0.00704	0.00833	0.01116
3	0.00906	0.00485	0.00446	0.00484	0.08020
6	0.00654	0.00352	0.01249	0.02363	0.06271
9	0.00896	0.00578	0.02430	0.01483	0.05272
12	0.00477	0.00507	0.00132	0.08638	0.08574
15	0.01014	0.01106	0.00701	0.02407	0.09272

Similar to the results obtained with TKAN previously, TKAT shows weaker predictive power on the single-step prediction task compared with a much simpler model, which may be explained by the excessive complexity of the model for this task. However, it appears that as soon as the model is used with only three steps ahead, it is able to offer much better performance than all the others. The results show that the results obtained are more stable during execution, and over the number of time steps with R^2 more than 50% higher than with the simple TKAN model. This makes sense, as the TFT-inspired transformer architecture is much more complete and considered state-of-the-art for these tasks.

Another very interesting result is the comparison between the model using the TKAN layer and the one based on LSTMs. While the architecture of the model makes the LSTM perform much better than on its own, it appears that the TKAN layer is able to improve performance by around 5% on average. One reading we can therefore make of the results here is that the overall architecture of the model plays an important role in the results obtained, particularly in comparison with the simple model variant, but that the TKAN layers are capable of increasing model performance even in more complex architectures.

4.2 Performance Metrics Summary of the TKAT versus variant closer to TFT

. The model, while inspired by the architecture of the temporal fusion transformer, presents some major divergences from it, due to the task we are studying. The first is the absence of static covariates, which are less relevant here, especially as the inclusion of some of them would have completely altered the training tasks themselves and thus made comparison with TKAN performance more difficult. Secondly, as we changed the whole upper part of the TFT after the multiple head, mainly because our tasks are very different from the ones they used, we felt that this architecture would not match our problem. This difference is mainly due to the fact that the data in the TFT article is data where the known part of the features is much larger, whereas our tasks have many more observed inputs. That said, we still tested variants closer to this article to see if our intuition on this subject was justified or not.

Table 3.3 – R^2 Average: TKAT variants closer to TFT

Time	TKAT-A	TKATN-A	TKAT-B	TKATN-B
1	0.29237	0.29502	0.28861	0.31119
3	0.19352	0.11464	0.16413	0.12965
6	0.11667	0.11588	0.14149	0.12075
9	0.07502	0.10003	0.09655	0.05756
12	0.05745	0.06382	0.07805	0.05968
15	0.04640	0.03845	0.05999	0.04139

Table 3.4 – R^2 Standard Deviation: TKAT variants closer to TFT

Time	TKAT-A	TKATN-A	TKAT-B	TKATN-B
1	0.04861	0.03480	0.08500	0.00833
3	0.02206	0.04145	0.02807	0.02665
6	0.02927	0.02070	0.01188	0.01567
9	0.01558	0.02216	0.01776	0.01475
12	0.01437	0.01354	0.01431	0.02162
15	0.01191	0.00834	0.01612	0.00937

In view of these results, it's more than obvious that using the full architecture would lead to poorer results for our specific tasks. The architecture we propose is therefore clearly different from that of TFT, but is probably not adapted to the same tasks by construction. What we want to highlight is that while the TFT inspiration gives excellent results for improving the performance of simple models, it needs to be adapted to specific learning tasks, as here the results show not only a lower mean R^2 but also much more variance in the results.

4.3 Number of parameters per model

A metric often looked at in Deep-Learning architectures is the number of parameters used by them. Here is the number of trainable parameters for the above models, calculated by TensorFlow 2.16.

Table 3.5 – Number of weights per model

Time	TKAT	TKAT non-KAN	TKAN	GRU	LSTM
1	1,022,275	1,061,461	99,426	97,001	128,501
3	1,027,077	1,066,263	99,628	97,203	128,703
6	1,043,280	1,082,466	99,931	97,506	129,006
9	1,070,283	1,109,469	100,234	97,809	129,309
12	1,108,086	1,147,272	100,537	98,112	129,612
15	1,156,689	1,195,875	100,840	98,415	129,915

Clearly, while TKAN's layers are not very different from standard RNNs such as GRU and LSTM, TKAT displays almost ten times as many parameters due to its more complex architecture. It's worth noting, however, that almost 65% of these trainable parameters belong to the variable selection part of the network, whereas the TKAN/LSTM layers only account for 10% of the total trainable weights. This metric, while interesting in some respects, should be taken with caution, as two models with the same number of parameters can have very different learning times, and the KAN part is, for the same number of parameters, more difficult to train than a standard MLP.

4.4 Results over Steps on 30 steps forecasting task

Finally, we also wanted to display the root-mean-square error for each forward step as a function of a model, all trained to predict the next 30 periods, because although the R^2 is readable, it gives no information on errors over the steps. All the models are those described above, with one extra, the MLP. This flattens the inputs received and simply applies two dense layers of 100 units with relu activation, before a final dense linear layer of 30 units to match the number of steps to be predicted. The models therefore have the following number of parameters:

Table 3.6 – Trainable Parameters Count

Model	Number of Trainable Parameters	R^2
TKAT	1,561,704	0.127743
TKAN	102,355	0.068004
MLP	298,230	-0.036020
GRU	99,930	0.055263
LSTM	131,430	-0.008245

The results were obtained after a single run for this graph.

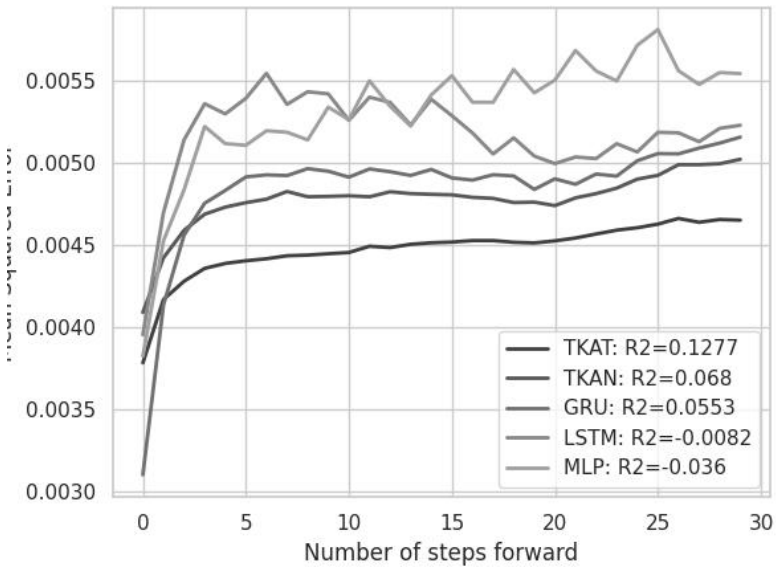


Figure 3.5 – Model and errors based on number of steps forward

The graph clearly shows the clear outperformance of GRU in the first step, in line with previous results, but also the fact that when there are more steps, TKAT and TKAN improve the quality of the forecast.

5 Conclusion

In this chapter, we have proposed an adaptation of two cutting-edge concepts in the field of deep learning, namely the Transformer architecture and the recent Kolmogorov-Arnold network. The obtained results show a real improvement over standard methods and demonstrate that TKAN has to be considered in Transformer-type architectures. With our specific learning task, different from that commonly applied for the Temporal Fusion Transformer, we show that the architecture needs to be modified "task by task" to achieve better performances. Our final methodology clearly outperforms the standard method, even though it lacks flexibility to use static covariates. It is also worth noting that this research has shown that the main architecture of the model plays a more important role than the single layer in performance. These results are very promising and confirm those we obtained with the TKAN model introduced in the previous chapter.

Part II

On Some Regime Switching Mechanism For Time series

Chapter 4

Deep State Space Recurrent Neural Networks for Time Series Forecasting

Abstract

The objective of this chapter is to study some neural networks architectures to model the dynamics of the cryptocurrency market. Traditional linear models often struggle to precisely capture its peculiar dynamics, in contrast to more advanced Deep Neural Networks (DNNs) that are widely known for their proficiency in time series forecasting. We introduce innovative neural network architectures that integrate the principles of econometric state space models with the dynamic capabilities of Recurrent Neural Networks (RNNs). Knowing there exist several hidden regimes, an encoder-decoder based switching mechanism is specified. Its performances are empirically tested, particularly its ability to assess the time varying transition probabilities between our hidden states.

Contents

1 Introduction

Digital assets constitute the most disruptive innovations of the last decade in finance. The primary intention of blockchain development was not to create a new currency, but to establish the principles of a functional decentralised cash payment system Procházka [Pro18]. The launch of bitcoin in 2008 Nakamoto [Nak09] was the wake-up call for the development of other crypto-currencies. Since the last decade, we have witnessed the birth of several thousand cryptocurrencies according to CoinMarketCap. Many still wonder if this is the emergence of a new asset class or just a bubble Garcia et al. [Gar+14] and Cheah and Fry [CF15].As this market has rapidly grown, so has the interest of researchers in this asset class. The particular characteristics of such assets have led some researchers to study their behaviour, in particular through statistical analysis and stochastic models on their returns [LAM22; IM21].

The digital asset market is young and has seen an exponentially rapid growth in recent years. Empirical studies show that this disruptive market has differentiated itself from other traditional financial markets by particular features: very high volatilities Katsiampa [Kat17], inverse leverage effects Garcia-Jorcano and Benito [GB20] and López-Martín, Arguedas-Sanz, and Muela [LAM22], skewed distributions, high kurtosis, etc. Despite the increasing interest in cryptocurrencies, it remains complicated to model the dynamics of their financial returns, mainly due to regularly observed periods of high volatility and alternating booms and bursts Katsiampa [Kat17]. A statistical study of bitcoin closing price distribution and dynamics cannot be induced by naive linear models, suggesting the existence of several market regimes. Cretarola and Figà-Talamanca [CF20] proposed an estimation procedure of all parameters based on the conditional maximum likelihood approach Andersen [And70] and Cretarola and Figà-Talamanca [CF21] and Hamilton filtering Hamilton [Ham89b]. In addition, deep neural networks have increasingly been used for time series modeling, showing better results compared to more classical approaches [Ran+18; AS19; Nik+19]. Some researchers have adopted a state-space approach using neural networks [KC18; Ilh+21]. There exists a rapidly growing literature on digital assets and deep learning applications to financial time series, particularly neural network models for forecasting purposes. The novelty of these assets sets them apart from conventional financial assets, particularly in the behaviour of their returns and volatility. These characteristics make the task of modeling them more complex. Linear econometric models, such as ARMA, find their limits due to a lack of flexibility and their difficulty in inducing certain empirical characteristics. The growing interest lies in the ability of neural network-based methods to approximate highly nonlinear functions. The focus on Recurrent Neural Networks (RNNs) for forecasting returns, volatilities, risk measures, and other quantitative metrics in finance and economics, is well-justified, due to their unique architecture capable of capturing time dependency effectively. RNNs are particularly well-suited to sequential data, which makes these models a relevant choice for time series analysis, which is common in financial markets. However, the characteristics of a time series may vary depending on the regime (boom or bust, e.g.) we are in. In this chapter, we introduce innovative neural network architectures that merge the principles of classic econometric state space models with RNNs. This is achieved by implementing a hidden switching

mechanism among multiple networks, where the transition probabilities vary over time and are influenced by certain observable covariates. In the next section, we recall the framework of regime switching models which will be used later to be confronted with deep learning models to estimate model parameters, including time varying transition probabilities. In section 4, we will specify some deep learning models. We will begin with some basic models and progressively incorporate more complexity, aiming to propose extensions and improvements to the current state-of-the-art models.

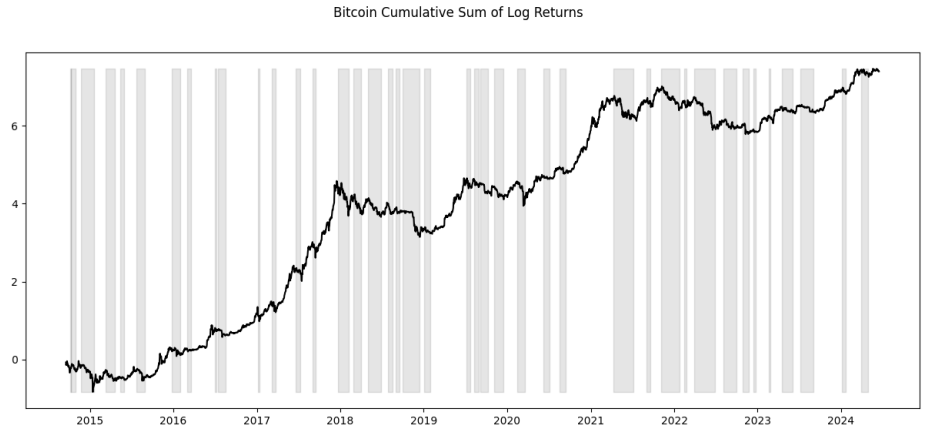


Figure 4.1 – Bitcoin cumulative sum of log returns. The shaded part of the figure represents the downward trends observed. We have defined a "bearish regime" as the period when the 20-day rolling average of the cumulative sum of log returns, shifted by 20 days, is lower than the current 20-day rolling average of the cumulative sum of log returns.

2 Simple Regime Switching models with Time Varying Transition Probabilities

Regime switching models are widely used by econometricians to model nonlinear dynamics of financial returns. These models have been introduced by Hamilton [Ham89b] in macroeconometrics. They are particularly relevant when certain time series exhibit distinct dynamics that depend on the state of the economy. We intend to apply the latter intuition to RNNs, particularly focusing on GRUs, LSTMs and TKANs. Our approach involves developing progressively more intricate models. We begin with GRUs, a specific type of RNN, and then enhance them by incorporating switching mechanisms. We will have the same approach for LSTMs and TKANs, introduced in the previous part of the thesis. Let us consider a (possibly non-homogeneous) Markov chain $(s_t)_{0\leq t\leq T}$. Here, $s_t \in \{1,\dots,m\}$ may be interpreted as the state of the economy (or "the market") at time t. We will consider $(s_t)_{0\leq t\leq n}$ as an irreductible chain, with associated transition probabilities $p_{ij,t} := \mathbb{P}(s_t = j | s_{t-1} = i, \mathcal{F}_{t-1})$, for any (i,j) in $\{1,\dots,m\}$ and any time t. We will study a discrete time series of closing prices $(Y_t)_{0\leq t\leq n}$. The associated log-returns are $r_t := \ln Y_t - \ln Y_{t-1}$. The series $(r_t)_{0<t\leq n}$ may most often be considered as strongly stationary for many financial assets (during reasonably long periods of time without any "structural break"), meaning that the joint distribution of the log returns denoted $(r_t, ..., r_{t-p})$ is independent of t

for all p. A basic regime switching model is typically written as

$$r_t = \alpha_{s_t} + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_{s_t}^2), \tag{4.1}$$

for some model parameters $(\alpha_1, \ldots, \alpha_m, \sigma_1, \ldots, \sigma_m)$. Markov Switching models are useful to capture switches across market regimes. The seminal model of Hamilton [Ham89b] assumed that the dynamics of the states (s_t) is purely exogenous and independent of the realizations of the variables of interest (r_t). Diebold, Lee, and Weinbach [DLW93] extended the model by assuming that the transitions between regimes could be caused by some underlying explanatory variables. The full amount of information that is available at the beginning of time t is denoted as $\mathcal{F}_{t-1}$. Typically, $\mathcal{F}_{t-1}$ records all returns and explanatory variables that have been observed before time t. Then, this induces a filtration $\mathcal{F} := (\mathcal{F}_t)_{t\geq 0}$. In a more general framework, we have to manage time varying transition probabilities across m states, justifying the notation

$$p_{ij,t} := \mathbb{P}(s_t = j | s_{t-1} = i, \mathcal{F}_{t-1}), \quad \sum_{j=1}^{m} p_{ij,t} = 1.$$

We denote by P_t the associated transition matrix for each time step $t \in \{0, \ldots, T\}$:

$$P_t := \begin{pmatrix} p_{11,t} & \cdots & p_{1m,t} \\ \vdots & & \vdots \\ p_{m1,t} & \cdots & p_{mm,t} \end{pmatrix}.$$

Our transition matrix will be defined as a measurable map of some past covariates, i.e. P_t belongs to $\mathcal{F}_{t-1}$. To be specific, for any $i \in \{1, \ldots, m\}$, let $(z_t^{(i)})_{t\geq 0}$ be a series of random vectors, where $z_{t-1}^{(i)} \in \mathcal{F}_{t-1}$ records the covariates that will be used to define the time varying transition probabilities from state i between the times $t-1$ and t. For notational convenience, we concatenate the vectors $z_t^{(i)}$, $i \in \{1, \ldots, m\}$ to build a new vector z_t. In terms of specification, for any $i, j \in \{1, \ldots, m\}$, $i \neq j$, and any t, we will set

$$p_{ij,t} := \frac{\exp(\beta_{i,j} z_{t-1}^{(i)})}{\sum_{k=1}^{m} \exp(\beta_{i,k} z_{t-1}^{(i)})}. \tag{4.2}$$

The latter time-varying transition probabilities depend on unknown (row) vectors of parameters $\beta_{i,j}$, $i, j \in \{1, \ldots, m\}$, $i \neq j$, that have to be estimated. The final vector of unknown parameters will be denoted as θ. It stacks the constants α_k, $k \in \{1, \ldots, m\}$ and the slope parameters $\beta_{i,j}$, $(i, j) \in \{1, \ldots, m\}^2$, $i \neq j$. To estimate θ, the maximum likelihood method is usually invoked. The associated log-likelihood function is given by

$$L_T(\theta) := \sum_{t=1}^{T} \log\left(f(r_t | \mathcal{F}_{t-1})\right), \tag{4.3}$$

Note that, under (4.1), we have

$$f(r_t|s_t) = \frac{1}{\sqrt{2\pi\sigma_{s_t}^2}}\exp\Big(-\frac{(r_t-\alpha_{s_t})^2}{2\sigma_{s_t}^2}\Big). \tag{4.4}$$

Note that the joint density of r_t and the unobserved variable s_t is

$$f(r_t, s_t|z_{t-1}) = f(r_t|s_t, z_{t-1})p(s_t|z_{t-1}), \tag{4.5}$$

with obvious notations. The marginal density of r_t is obtained by summing over all the possible states:

$$\begin{aligned} f(r_t|\mathcal{F}_{t-1}) &= \sum_{s_t=1}^{m} f(r_t, s_t|z_{t-1}), \\ &= \sum_{s_t=1}^{m} f(r_t|s_t, z_{t-1})p(s_t|z_{t-1}). \end{aligned} \tag{4.6}$$

Using (4.6) we can rewrite the likelihood as

$$L_T(\theta) := \sum_{t=1}^{T} \log\Big(\sum_{s_t=1}^{m} f(r_t|s_t, z_{t-1})p(s_t|z_{t-1})\Big). \tag{4.7}$$

In the case of an autoregressive model of order p with an underlying hidden first-order Markov chain (called MSAR(p)), we would write the density of r_t given the past information contained in z_{t-1}, we would need to consider s_t as well as s_{t-1}. As mentionned before, s_t is unobserved, hence to solve this issue we would consider the join density of r_t, s_t and s_{t-1}.

$$f(r_t, s_t, s_{t-1}|z_{t-1}) = f(r_t|s_t, s_{t-1}, z_{t-1})p(s_t, s_{t-1}|z_{t-1}) \tag{4.8}$$

$f(r_t|z_{t-1})$ can be computed by summing the possible values of s_t and $s_t - 1$ as follows:

$$\begin{aligned} f(r_t|z_{t-1}) &= \sum_{s_t}^{m}\sum_{s_{t-1}}^{m} f(r_t, s_t, s_{t-1}|z_{t-1}), \\ &= \sum_{s_t}^{m}\sum_{s_{t-1}}^{m} f(r_t|s_t, s_{t-1}, z_{t-1}) \end{aligned} \tag{4.9}$$

$$p(s_t, s_{t-1}|z_{t-1})$$

where $p(s_t, s_{t-1}|z_{t-1}) = p(s_t|s_{t-1}, z_{t-1})p(s_{t-1}|z_{t-1})$. To make prediction, one has to evaluate

$$\hat{r}_t = E[r_t|\mathcal{F}_{t-1}] = \sum_{j=1}^{m} P(s_t = j|\mathcal{F}_{t-1})E[r_t|\mathcal{F}_{t-1}, s_t = j].$$

2.1 Simple Regime Switching (2 states)

To proceed, we considered the daily log returns $r_t := \ln P_t - \ln P_{t-1}$ of Bitcoin. Initially, we estimate a Markov switching model with constant transition probabilities in order to determine whether there exist two or rather three underlying states. Next, we take the analysis further by including covariates and time varying transition probabilities in the model. This allows us to understand how these covariates influence the likelihood of switching between different states, as captured by the transition matrix. Furthermore, by analyzing our time-varying transition matrices, we can assess to what extent the model's behaviour remains persistent over time. You will find below the results of the estimations of Model (4.1) with two market regimes. The results with three market regimes are available in the appendix 7.1-7.2.

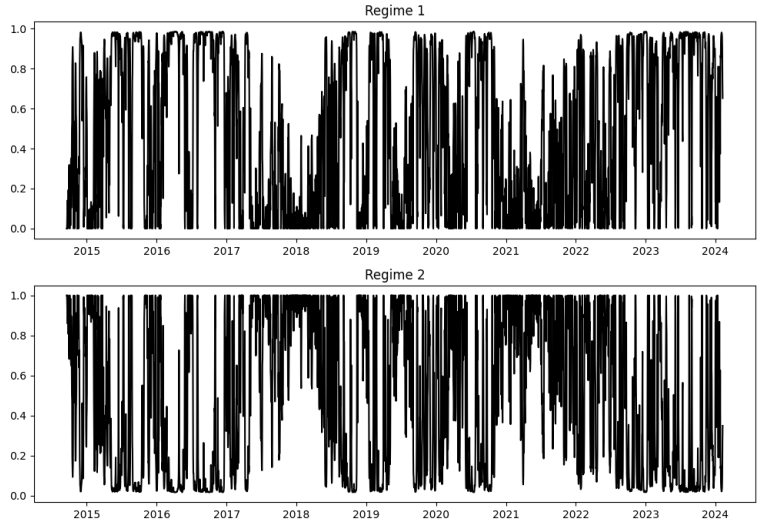


Figure 4.2 – Smoothed Marginal Probabilities (basic MS model, 2 regimes)

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
constant	0.0014	0.001	2.677	0.007	0.000	0.003
variance	0.0002	5.15e-05	4.192	0.000	0.000	0.000
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
constant	0.0015	0.001	1.107	0.268	-0.001	0.004
variance	0.0024	0.000	11.890	0.000	0.002	0.003
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
p[1->1]	0.8643	0.051	17.028	0.000	0.765	0.964
p[2->1]	0.1479	0.049	3.033	0.002	0.052	0.244

Table 4.1 – Markov Switching Model Results (basic MS model, 2 regimes)

The table 4.1 shows the model's estimated coefficients, their associated standard errors, and

p-values for a two-regimes non-time-varying Markov switching. For instance, in state 1, the constant term's coefficient is noted as 0.0014, with a standard error of 0.001 and a p-value of 0.007, indicating a statistically significant difference from zero at the 1% significance level. Similarly, the coefficient for the probability of remaining in state 1 is reported as 0.8643, with a standard error of 0.051 and a p-value practically equal to zero, suggesting a highly significant estimate that challenges the hypothesis of a coefficient value of 1 at the 0.1% level. In state 2, the constant term coefficient is noted as 0.0015 with a p-value of 0.268, meaning the null hypothesis cannot be rejected: the constant term is not statistically different from zero. The relatively high values of $p[1 \to 1]$ and $p[2 \to 2]$ prove the tendency for the system to remain in its current state rather than rapidly transitioning to another one, which is a sign of stability of the model. Finally, we observe a large difference in terms of conditional variances between regime 1 and regime 2 (0.0002 vs 0.0024, respectively). Cryptocurrency markets are then inherently volatile and susceptible to break.

The analysis of similar results in the case of three regimes (Figure 7.1) shows a different and more puzzling picture. The first two regimes are highly volatile, with frequent switches between them. Due to low means and volatility, Regime 1 appears rather artificial and spurious. The third is more clear-cut, since it is very persistent and exhibits a high level of volatility. Thus, in terms of interpretation and financial intuition, a two-state MS model seems to be more realistic.

2.2 Covariates

For each model, we first estimate smooth marginal probabilities. In our analysis, we would consider two states, $s_t \in \{0, 1\}$. Our prior is that one state is related to normal return and the other is related to a state with more agitation. We assume :

$$r_t = \begin{cases} a_1 + \sigma_1 \epsilon_t, & \text{if } s_t = 1. \\ a_2 + \sigma_2 \epsilon_t, & \text{if } s_t = 2. \end{cases} \tag{4.10}$$

Where $a_2 \geq a_1$. We expect that $\sigma_2 > \sigma_1$. To compute the time-varying transition probabilities, we used the following covariates: High Minus Low (HML) and an intraday variance indicator denoted (IV) following the methodology of Inzirillo and Mat [IM21]. In this paper, they show the impact of such factor and also its capacity to sum up the information in a subspace using statistical techniques to reduce the dimension of the input data. We define $\theta_t^i \in R_+^4$, the set of parameters for the i-th asset. $\theta_t^i := \{O_t^i, H_t^i, L_t^i, C_t^i\}$. The proxy of intraday volatility of the asset is given by,

$$f_t(\theta_t^i) := \Psi(H_t^i, L_t^i), \tag{4.11}$$

where

$$\Psi = \begin{cases} \log\left(\frac{H_t^i}{L_t^i}\right), & \text{if } O_t^i \geq C_t^i. \\ \log\left(\frac{L_t^i}{H_t^i}\right), & \text{if } O_t^i < C_t^i. \end{cases}$$

and we obtain,

$$IV_t^i = f_t(\theta_t^i)^2 \tag{4.12}$$

For deep learning tasks, if needed, after constructing our vector of covariates for each date t, we apply standardization on our dataset in order to proceed to the estimation of the model parameters for the different market regimes.

2.3 Impact of Covariates

This section explores how additional information (covariates) affect the likelihood of market regime changes. We are interested in seeing if these covariates help us identify different market regimes. Initially, we tested a model with two volatility regimes (a regime with high volatility and another one with low volatility), in terms of asset returns. First, we study time-varying transition probabilities using a "High-Minus-Low" (HML) indicator as a covariate. Subsequently, we extend our analysis by integrating other indicators, notably the "intraday volatility" (IV, see (4.12)), to assess their influence on the accuracy of our estimated parameters. Our goal is to analyze the dynamics of the time-varying transition matrices when such factors are used, and to verify whether their integration induces significantly different interpretations. Table 4.2 displays the results using HML only. We clearly still identify the presence of two distinct regimes differentiated by their conditional variances. The introduction of HML inside the model as a covariate for the estimation of time-varying transition probabilities makes sense. The new corresponding coefficients are statistically different from zero, indicating a real effect of HML on transition probabilities. When we enrich the model with IV (see Figure 4.4 and Table 4.3), the picture does not change, and we conclude that using HML and IV as drivers of time-varying transition probabilities is meaningful. The same experiment with three regimes (Table 4.3) provides a less clear-cut view, as without covariates. In particular, many estimated coefficients related to HML and/or IV are no longer statistically different from zero, casting doubt on model specification. In view of the latter results, we preferred to consider two states for the model, where state 2 is related to relatively more volatile asset returns than state 1.

Regime Switching TVTP with HML Factor

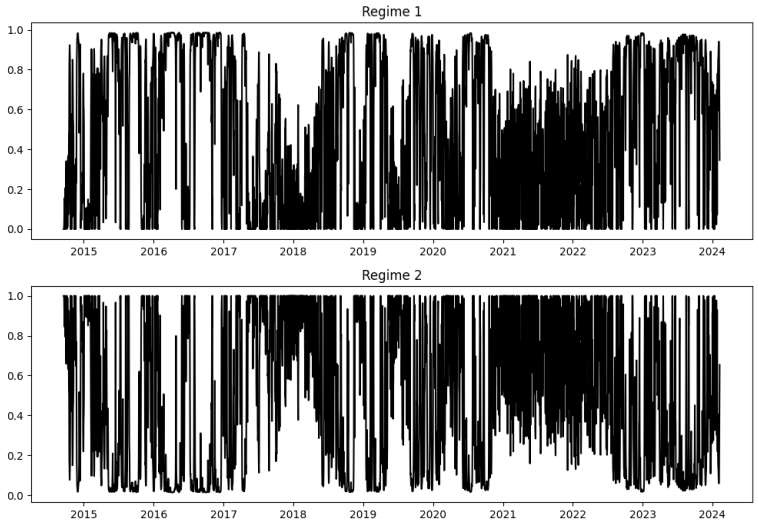


Figure 4.3 – Smoothed Marginal Probabilities

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0013	0.001	2.552	0.011	0.000	0.002
sigma2	0.0002	5.13e-05	3.817	0.000	9.53e-05	0.000
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0016	0.001	1.190	0.234	-0.001	0.004
sigma2	0.0025	0.000	10.552	0.000	0.002	0.003
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
p[1->1].const	1.2185	0.439	2.775	0.006	0.358	2.079
p[2->1].const	-1.3992	0.258	-5.424	0.000	-1.905	-0.894
p[1->1].hml	-1.2059	0.236	-5.115	0.000	-1.668	-0.744
p[2->1].hml	0.2336	0.111	2.105	0.035	0.016	0.451

Table 4.2 – Markov Switching Model Results, Figure 4.3

Regime Switching TVTP with HML and IV Factors

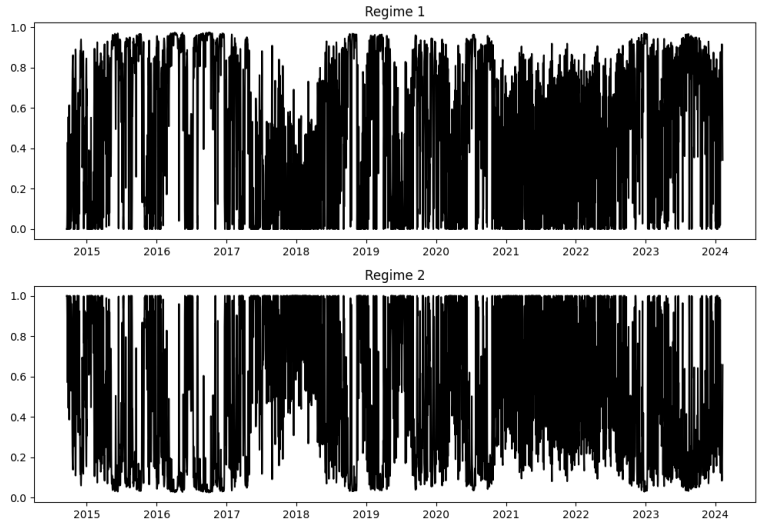


Figure 4.4 – Smoothed Marginal Probabilities

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0012	0.000	2.777	0.005	0.000	0.002
sigma2	0.0002	2.67e-05	6.553	0.000	0.000	0.000
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0017	0.001	1.353	0.176	-0.001	0.004
sigma2	0.0025	0.000	13.731	0.000	0.002	0.003
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
p[0->0].const	-0.1682	0.456	-0.369	0.712	-1.062	0.726
p[1->0].const	-0.3797	0.259	-1.467	0.142	-0.887	0.128
p[0->0].hml	-0.8880	0.312	-2.848	0.004	-1.499	-0.277
p[1->0].hml	0.4328	0.146	2.956	0.003	0.146	0.720
p[0->0].iv	-1.5306	0.502	-3.049	0.002	-2.514	-0.547
p[1->0].iv	-0.8256	0.192	-4.304	0.000	-1.202	-0.450

Table 4.3 – Regime Switching Parameters (HML,IV), Figure 4.4

3 The Evolution of Deep Learning Methods

In this section, we review the fundamental architectures of neural networks, from the first basic attempts to the most advanced models as documented in the literature. Afterwards, we introduce our innovative state space neural network architecture, with a new switching mechanism. We also proposed a new type of layer for neural networks combining the power of RNNs with the efficiency of attention-free transformer architecture called the AF-LSTM layer.

3.1 Feed Forward Networks (FNNs)

The first stage of the story is based on the concept of MP Neuron Mcculloch and Pitts [MP43]. Such MP neurons were originally used for classification tasks. They take binary inputs and return a binary output according to a trigger.

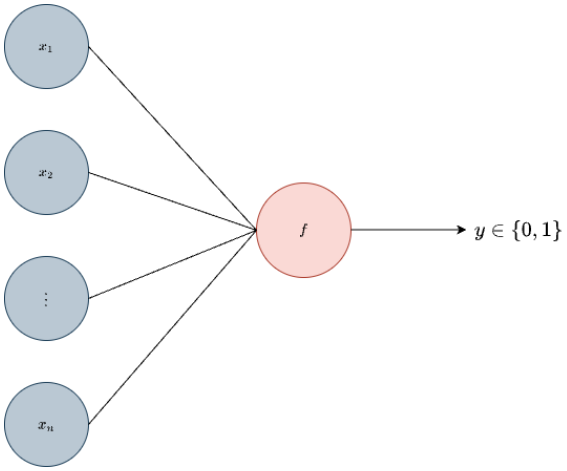


Figure 4.5 – MP neuron

Figure 4.5 shows the structure of a basic MP neuron, composed by an input vector X and a function f which takes two values 0 or 1. If the sum of the x_i, $i \in \{1, \ldots, n\}$, is higher than a trigger b, then the value of the output y is set to 1, and 0 otherwise. Mathematically, we can write the MP neuron predictor as follows:

$$f(x) := \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i \geq b. \\ 0 & \text{otherwise.} \end{cases}$$

Perceptron

Perceptrons, introduced by Rosenblatt [Ros58] are an extension of the previous MP neurons Mcculloch and Pitts [MP43] that can take any real value as input. Now, each element of the input vector X is multiplied by a weight. Moreover, contrary to MP neurons, one or several layers are usually added in the network. This significantly increases the flexibility of the model. The addition of a vector of parameters θ introduced the error-correction learning in the neural network and made it possible to adjust the weights to improve the classification task.

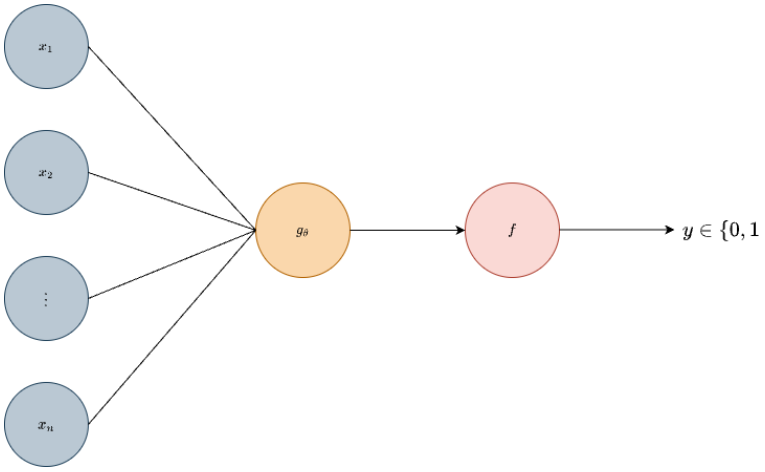


Figure 4.6 – Perceptron

In Figure 4.6, g_θ is a parametrized function which takes an input $\{x_i\}_{i=1}^n$ where x_i is a real number for every i. In a second stage, the output of g_θ will be given as inputs of $f \in F$, where F is a set of "activation functions" which yield the final decision and return a binary variable (in the case of classification). Mathematically, this can be written as follows:

$$f(g_\theta(x)) := \begin{cases} 1 & \text{if } g_\theta(x) \geq b. \\ 0 & \text{otherwise.} \end{cases}$$

Note that the threshold b is a learning parameter that has to be set during the training stage of the network. Despite the power of MP neurons and perceptrons, they cannot easily manage nonlinearity in problem solving. Although Perceptrons have an embedded layer, their amount of flexibility is limited. To solve more complex nonlinear problems, Multi Layer Perceptron (MLP) has been introduced. These architectures are a cornerstone of modern neural network research and applications.

Multi Layer Neural Networks

Let us define some notations: W_1 denotes the weights of the input layer, the first layer of the network. W_{h_p}, and $p \in [0, ..., n]$ denotes the weight of the hidden layers and $n-1$ the number of total hidden layers. W_{h_n} the vector of weight of the final hidden layer is then output layer of the network. $\sigma(\cdot)$ is the activation function of the network. In this case, we will use the logistic function:

$$\sigma(z) = \frac{1}{1 + \exp(-z)},$$

step after step calculation is given by the following equations,

$$\begin{aligned} h_1 &= \sigma(W_{h_0} x + b_{h_0}), \\ h_2 &= \sigma(W_{h_2} h_1 + b_{h_1}), \\ o &= \sigma(W_{h_3} h_2 + b_{h_2}). \end{aligned} \tag{4.13}$$

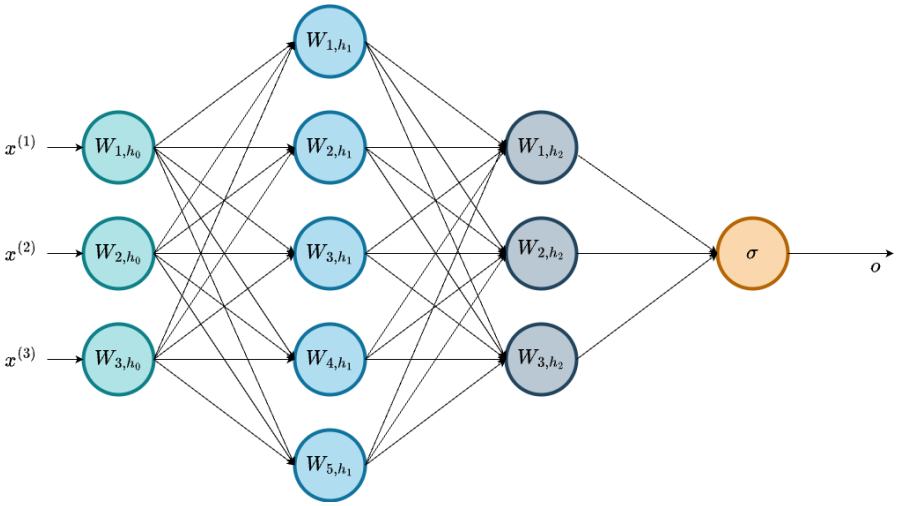


Figure 4.7 – MLP

As observed in Figure 4.7, an MLP [HSW89; Hay98; Cyb89] is an acyclic graph, oriented in one direction (from left to right, here). The layers are fully connected to each other. However, there is no connection between nodes within a layer. FFNs process information layer by layer without feedback connections. The missing feedback connections make them unsuitable for sequential data. FFNs process each element of the input independently, treating it as an isolated piece of information. These networks are unable to consider context or previous values, which are essential for understanding sequential data such as time series. Nonetheless, perceptrons have yielded basic building blocks for a lot more complex and flexible predictors such recurrent neural networks.

3.2 Recurring Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) have been proposed to address the "persistence problem", i.e. the potentially long-term dependencies between the successive observations of some time series. Here, an iterative process inside every cell allows information to persist through time and brings consistency with respect to temporal dependency. Therefore, RNNs most often outperform "static" networks such as MLPs [LJH15]. Gated Recurrent Units (GRU) (Figure 4.8) were proposed by Cho et al. [Cho+14a]. They constitute the building blocks of some family of RNNs that explicitly take into account time ordering. A GRU embeds two "gated" mechanisms called "reset" and "update" detailed equations (4.14).

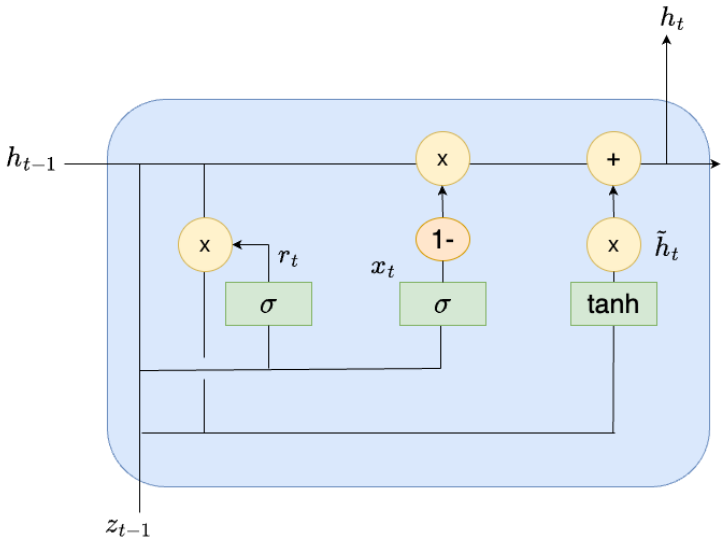


Figure 4.8 – GRU

$$
\begin{aligned}
x_t &= \sigma(W_z[h_{t-1}, z_{t-1}]) \\
r_t &= \sigma(W_r[h_{t-1}, z_{t-1}]) \\
\tilde{h}_t &= \tanh(W[r_t \odot h_{t-1}, z_{t-1}]) \\
h_t &= (1 - a_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
\tag{4.14}
$$

Traditional methods of gradient descent may not be sufficiently effective for training Recurrent Neural Networks (RNNs), particularly in capturing long-term dependencies [BSF94]. Meanwhile, Chung et al. [Chu+14] conducted an empirical study revealing the effectiveness of gated mechanisms in enhancing the learning capabilities of RNNs. Actually, RNNs have proved to be one of the most powerful tools for processing sequential data and solving a wide range of difficult problems in the fields of automatic natural language processing, translation, image processing, and time series analysis. Researchers have been able to mimic human abilities like selectively focusing on crucial information, similar to how our attention works on input sequences. This innovative mechanism will be discussed in a later section.

Long-Short Term Memory (LSTM)

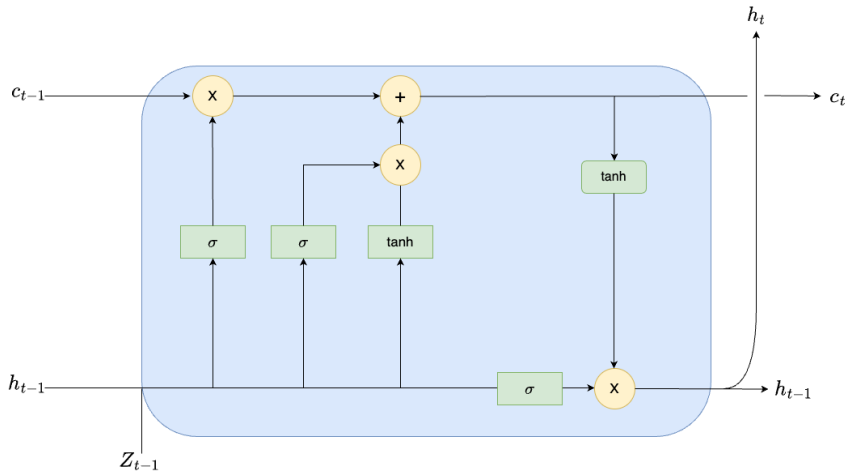


Figure 4.9 – LSTM Cell

Long-Short Term Memory (LSTM) networks [CD19] are a specific type of RNNs with gated mechanisms. Like GRUs, LSTMs seek to control the flow of information through some gates without having to use a memory unit. However, GRUs have only two gated mechanisms whereas the LSTM cell has three: the input gate, forget gate, and update gate. One key particularity of LSTM architecture is that the update and forget gates are separated, which makes LSTMs more complex and evolved than GRUs. They have been designed to avoid the "vanishing gradient" problem. The latter problem often appears during the update of the usual RNN model proportionally weighted to the loss partial derivatives. Sometimes, the gradients of error terms may be vanishingly small and weights may not be updated during the learning task. To be specific, the LSTM cell built on Figure 4.9 is defined by the following equations:

$$\begin{aligned} f_t &= \sigma(W_f[h_{t-1}, z_{t-1}] + b_f) \\ i_t &= \sigma(W_i[h_{t-1}, z_{t-1}] + b_i) \\ \tilde{c}_t &= \tanh(W_c[h_{t-1}, z_{t-1}] + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ o_t &= \sigma(W_o[h_{t-1}, z_{t-1}] + b_o) \\ h_t &= o_t \odot \tanh(c_t), \end{aligned} \tag{4.15}$$

where i_t, f_t and o_t denote the input, forget, and output gate, respectively. Our set of parameters, denoted by $\theta := \{W_f, W_i, W_c, W_o, b_f, b_i, b_c, b_o\}$, stacks the weights and intercepts of the model and $z_{t-1} \in \mathbb{R}^{s_{len} \times d}$ is the input vector at time t. s_{len} denotes the length of the input sequence and d the number of features. The notation $g(z_{t-1}; \theta) := \text{LSTM}(z_{t-1}; \theta)$ represents the sequence of operations performed by the LSTM on z_{t-1} with parameters θ. In the case of financial time series prediction, we seek to predict future returns, volatilities, etc., based on the history of past returns. For instance, $\mathbb{E}[r_t|\mathcal{F}_{t-1}]$ would be estimated by a linear transformation of the LSTM outputs, i.e. by $\hat{y}_t := W_y g(z_{t-1}; \theta) + b_y$, for some parameters W_y and b_y.

4 Deep State Space Models

4.1 Switching Mechanism

In this subsection, we propose a novel approach to estimate switching probabilities through neural networks. Let us consider a hidden Markov chain $(s_t)_{0\leq t\leq T}$. Here, $s_t \in \{1,\dots,m\}$ may be interpreted as the market regime at time t. We will consider $(s_t)_{0\leq t\leq n}$, an irreducible chain, with the associated conditional transition probabilities $\mathbb{P}(s_t = j|s_{t-1} = i, \mathcal{F}_{t-1})$, for any (i,j) in $\{1,\dots,m\}$ and any time t and for some user-defined filtration $(\mathcal{F}_t)_{t\geq 0}$. Hamilton [Ham89b] assumed that the dynamics of the states (s_t) is purely exogenous and independent of the realizations of the variables of interest $r_t := \log(p_t) - \log(p_{t-1})$. Diebold, Lee, and Weinbach [DLW93] extended the model by assuming that the shifts between different states or regimes may be influenced by some underlying factors or explanatory variables. By default, $\mathcal{F}_{t-1}$ denotes the whole set of information accessible at the start of time t (asset returns, volumes, general purpose market information, etc). The proposed switching mechanism allows us to estimate the conditional probability of being in a given state for each time step t

$$\pi_{i,t|t-1} = \mathbb{P}(s_t = i|\mathcal{F}_{t-1}) \quad i \in \{1,...,m\}, \tag{4.16}$$

where m denotes the total number of market regimes. First, introduce the σ-algebra induced by a time series of random vectors $(\mathcal{Z}_j)_{j\leq t}$, i.e. $\mathcal{F}_{\mathcal{Z},t-1} = \sigma(\mathcal{Z}_t, \mathcal{Z}_{t-1}, \dots)$, and the associated filtration $(\mathcal{F}_{\mathcal{Z},t})_{t\geq 0}$. Typically, $\mathcal{Z}_t$ will be the vector obtained from a neural network that will be built from some financial time series (including quotes, volumes, bid-ask spreads, or other market information possibly) until and including $t-1$. In particular, $\mathcal{Z}_t$ is $\mathcal{F}_{t-1}$-measurable. We now assume that $\mathcal{F}_{\mathcal{Z},t-1}$ brings sufficient information to evaluate the conditional probabilities $\pi_{i,t|t-1}$, i.e.,

$$\pi_{i,t|t-1} = \mathbb{P}(s_t = i|\mathcal{F}_{\mathcal{Z},t-1}), \quad i \in \{1,...,m\}. \tag{4.17}$$

The latter probabilities $\pi_{i,t|t-1}$ will be estimated, computed recursively, and updated at each time step using the (conditional) transition probabilities

$$p_{ij,t} = \mathbb{P}(s_t = j|s_{t-1} = i, \mathcal{F}_{\mathcal{Z},t-1}). \tag{4.18}$$

To be specific, we will focus on the particular specification

$$\mathcal{Z}_t = W_{\mathcal{Z}} o_t + b_{\mathcal{Z}}, \tag{4.19}$$

where $W_{\mathcal{Z}} \in \mathbb{R}^{m\times o_{dim}}$ and $o_t \in \mathbb{R}^{o_{dim}}$. Here, $o_t := NN(\nu_{t-1})$ denotes the output of a neural network. The ReLU (Rectified Linear Unit) activation function applied to the output of the neural network helps to filter the important information extracted from market information. The input of the neural networks $\nu_{t-1} \in \mathbb{R}^d$ is a stacked vector of some "covariates" that are observable at $t-1$. Among the latter variables, we have not considered the past returns: in the current model, the transition probabilities are influenced by the behaviour of covariates only.

For each $t \in \{1, ..., T\}$, we compute our transition probabilities to build our transition matrix

$$P_t := \begin{pmatrix} p_{11,t} & \cdots & p_{1m,t} \\ \vdots & & \vdots \\ p_{m1,t} & \cdots & p_{mm,t} \end{pmatrix}. \tag{4.20}$$

At time t, our time varying transition probability matrix $P_t = P_{t-1}\rho_t$ provides a way of updating the transition matrix with some pieces of past information. To update the transition matrix P_t, we propose here to assume

$$\rho_t = \exp(\mathcal{Z}_t), \tag{4.21}$$

where the exponential map is applied componentwise. ρ_t will be subject to the transformation from a 1D vector to a 2D square matrix where rows and columns is associated to a market regime. Moreover, we can impose that the diagonal elements of ρ_t are one.

A true transition matrix P_t is obtained from

$$P_t = \text{softmax}(P_{t-1} \odot \rho_t), \tag{4.22}$$

where $\odot$ denotes componentwise multiplication. The SoftMax activation function is applied on every row of $P_{t-1} \odot \rho_t$.

As a second approach, it is tempting to enrich the latter way of estimating the latent states given some market information. Indeed, restricting ourselves to some "covariates" only may be questionable. Thus, we would like to add more information in the previous conditioning σ-algebra $\mathcal{F}_{\mathcal{Z},t-1}$. Typically, at time t, having a value of the t-th return (or the t-volume, etc.) has to improve the prediction of s_t. We particularize the additional stream of information that is induced by a sequence of random vectors $(y_t)_{t\geq 0}$. At the beginning of time t, the available information $\mathcal{F}_{t-1}$ includes all $\mathcal{Z}_{t-k}$, $k \geq 0$, all y_{t-j}, $j \geq 1$, and possibly other past market information. The new quantity of interest is denoted $\pi_{i,t|t}$ for each time step, with $\pi_{i,t|t} := \mathbb{P}(s_t = i|\mathcal{F}_t)$. The new conditional probabilities $\pi_{i,t|t}$ will be calculated by applying a type of Hamilton filter. Indeed, denoting $\tilde{\pi}_{i,t|t-1} := \mathbb{P}(s_t = i|\mathcal{F}_{t-1})$, we have

$$\begin{aligned} \pi_{i,t|t} = \mathbb{P}(s_t = i|\mathcal{F}_t) &\simeq \frac{\mathbb{P}(s_t = i, y_t|\mathcal{F}_{t-1})}{f(y_t|\mathcal{F}_{t-1})} \\ &= \frac{f(y_t|s_t = i, \mathcal{F}_{t-1})\mathbb{P}(s_t = i|\mathcal{F}_{t-1})}{\sum_{k=1}^m f(y_t|s_t = k, \mathcal{F}_{t-1})\mathbb{P}(s_t = k|\mathcal{F}_{t-1})} \\ &= \frac{f(y_t|s_t = i, \mathcal{F}_{t-1})\tilde{\pi}_{i,t|t-1}}{\sum_{k=1}^m f(y_t|s_t = k, \mathcal{F}_{t-1})\tilde{\pi}_{k,t|t-1}}. \end{aligned} \tag{4.23}$$

To justify the first identity, we implicitly assumed that y_t is the single additional piece of information between $t-1$ and t, for the purpose of state forecasting. In particular, $\mathcal{Z}_{t+1}$ does not matter. Moreover, assume that $\mathbb{P}(s_t = j|s_{t-1} = i, \mathcal{F}_{t-1}) = \mathbb{P}(s_t = j|s_{t-1} = i, \mathcal{F}_{\mathcal{Z},t-1}) = p_{ij,t-1}$. Thus, this yields

$$\tilde{\pi}_{k,t|t-1} = \sum_{l=1}^m p_{kl,t-1}\pi_{l,t-1|t-1}. \tag{4.24}$$

Moreover, $f(y_t|s_t = i, \mathcal{F}_{t-1})$ can be evaluated as

$$f(y_t|s_t = i, \mathcal{F}_{t-1}) = \phi((y_t - \hat{y}_{i,t})/\sigma_{i,t}),$$

where ϕ is the density of a $\mathcal{N}(0,1)$. Here, every quantity $\hat{y}_{i,1}$, $i \in \{1, \dots, m\}$, is an estimator of the conditional expectation of y_t given its state. The latter quantities have been obtained by some neural networks. The quantity $\sigma_{i,t}$ is the standard deviation of the quantities $\hat{y}_{i,1}, \dots, \hat{y}_{i,t-1}$. In other words, we assumed the law of the explained variable y_t is Gaussian, given its current state. This yields

$$\pi_{i,t|t} = \mathbb{P}(s_t = i|\mathcal{F}_t) = \frac{f(y_t|s_t = i, \mathcal{F}_{t-1}) \sum_{l=1}^{m} p_{kl,t-1}\pi_{l,t-1|t-1}}{\sum_{k,l=1}^{m} f(y_t|s_t = k, \mathcal{F}_{t-1}) p_{kl,t-1}\pi_{l,t-1|t-1}}. \tag{4.25}$$

Finally, (5.37) allows us to recursively calculate the quantities $\tilde{\pi}_{i,t|t}$ and then the $\pi_{i,t|t-1}$ by (4.24).

The quantities $\hat{y}_{i,t}$ refer to the output of Neural Networks (NNs) within the Switching NN architecture, which will produce probabilities as an ouput stored in $\pi_t \in R^m$, for each time step.

During the learning task, detailed in the next section, we will apply this methodology on different recurrent neural networks. The objective is to assess the capacity of prediction by drawing a backtest to evaluate which models perform the best.

4.2 Switching RNNs

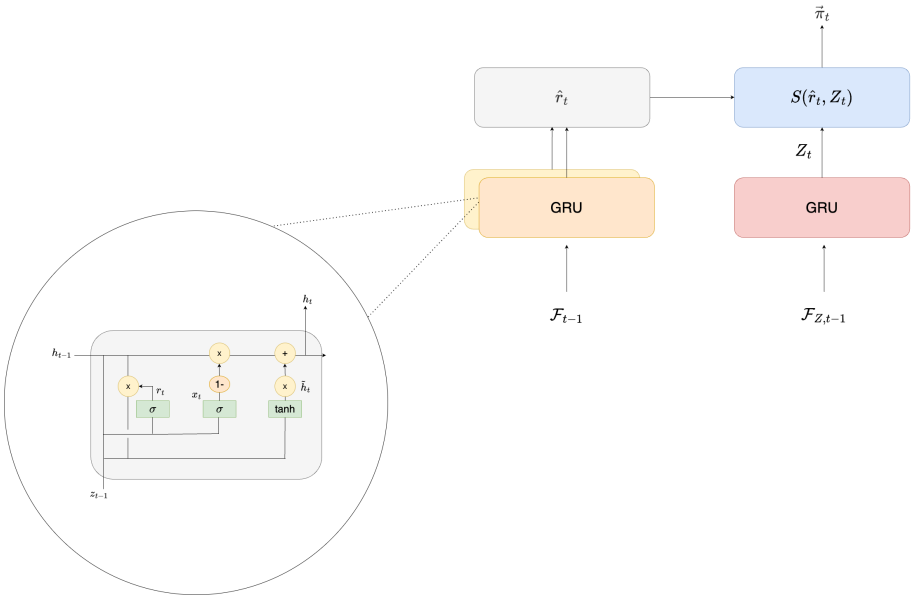


Figure 4.10 – Structure of m-GRU see (4.28)

In the previous section, we have introduced the switching mechanism used to estimate transition probabilities and therefore the probability of being in a given state. Now, we will explore its application to several architectures. We will start with GRUs, known for its ability to efficiently capture long-term dependencies in sequences. We will see how this mechanism can help the GRU to better adapt to the different dynamics present in the covariates. Secondly, we will apply this mechanism to LSTMs known for their ability to manage short and long-term memory. Finally, we will apply this mechanism to the recently introduced TKAN architecture, which combines the use of KANs [Liu+24] and memory management. The Recurrent Neural Networks (RNNs) are a specific type of neural networks architecture oriented along a temporal sequence. This network architecture is distinguished from others by the presence of a memory effect, allowing it to process sequential data effectively. One popular RNN variant is the Gated Recurrent Unit (GRU), known for its ability to capture long-term dependencies in sequences. The Switching GRU extends the standard GRU by taking into account a regime variable k in the update, reset and hidden state gate equations:

$$\begin{aligned} z_t^{(k)} &= \sigma(W_{k,z} \cdot [h_{k,t-1}, z_{t-1}] + b_{k,z}), \\ r_{k,t} &= \sigma(W_{k,r} \cdot [h_{k,t-1}, z_{t-1}] + b_{k,r}), \\ \tilde{h}_{k,t} &= \tanh(W[r_{k,t} \odot h_{k,t-1}, z_{k,t-1}]), \\ h_{k,t} &= (1 - z_{k,t}) \odot h_{k,t-1} + z_{k,t} \odot \tilde{h}_{k,t}. \end{aligned} \tag{4.26}$$

This update in the model allows modulating its behavior according to the current regime, providing increased flexibility. The introduction of this regime-switching mechanism in the GRU architecture is suitable for tasks where the dataset is subject to context variations, which is often the case in time series and particularly in highly volatile assets. By adapting its dynamics to the current regime, the Switching GRU can better handle the complex underlying patterns present in such data. The succession of operations detailed in (4.26) is used to estimate log returns of each possible state. On the right side of Figure 4.10 we have another GRU. We will denote $\text{GRU}_{k,in}$ the successive equations (4.26), and the other GRU (on the right side) will be denoted GRU_c which will be fed using the covariate only and not the entire dataset containing covariates and past observed values of the target. The GRU_c will transform covariates to Z_t vector such

$$Z_t = \text{GRU}_c(\mathcal{F}_{Z,t-1}) \tag{4.27}$$

The Z_t and the vector of $\vec{r_t} = (r_{1,t}, r_{2,t}, ..., r_{m,t})$ obtained previously with the use of $\text{GRU}_{k,in}$ will be the input of our switching mechanism detailed 4.1 to estimate the vector $\vec{\pi_t} = (\pi_{1,t}, \pi_{2,t}, ..., \pi_{m,t})$. The output of the framework is $\vec{\pi_t}$. In our regime-switching model, the function $S(r_t, Z_t)$ generates a vector of probabilities π_t for m regimes at time t. For a two-regime system, $\pi_t = (\pi_{t,1}, \pi_{t,2})$. The predicted regime $\hat{y}_t$ is determined by $\hat{s}_t = \arg\max_k(\pi_{t,k})$, which can be expressed by an indicator function $I_{t,k}$. The true regime $s_t \in \{0, 1\}$ is one-hot encoded. The confusion matrix C is then constructed as $C_{ij} = \sum_t I(\hat{s}_t = i \text{ and } s_t = j)$, where $\hat{s}_t$ is the predicted regime and s_t what we defined as a true regime.

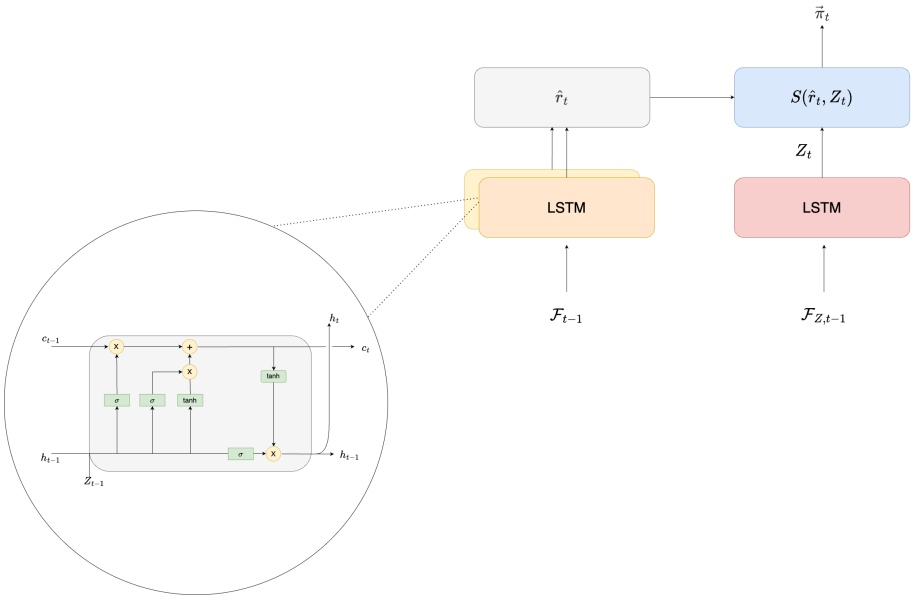


Figure 4.11 – Structure of the m-LSTM

The methodology would be the same for the LSTM [HS97a]. m-LSTM are stacked with the following ouputs $\{h_{k,t}\}_{k=1}^{m} := \{h_{1,t}, h_{2,t}, ..., h_{m,t}\}$ and a set of parameters $\{\theta_k\}_{k=1}^{m} := \{\theta_1, \theta_2, ..., \theta_m\}$ where;

$$\theta_k := \{W_{k,f}, W_{k,i}, W_{k,c}, W_{k,o}, b_{k,f}, b_{k,i}, b_{k,c}, b_{k,o}\},$$

Our model, described in Figure 4.12 shows the LSTM stacked on the left side. We introduce the notation $\text{LSTM}_{k,in}$ to denoted the succession of the following operation,

$$\begin{aligned}
f_{k,t} &= \sigma(W_{k,f} \cdot [h_{k,t-1}, z_{t-1}] + b_{k,f}), \\
i_{k,t} &= \sigma(W_{k,i} \cdot [h_{k,t-1}, z_{t-1}] + b_{k,i}), \\
\tilde{c}_{k,t} &= \tanh(W_{k,c} \cdot [h_{k,t-1}, z_{t-1}] + b_{k,c}), \\
c_{k,t} &= f_{k,t} * c_{k,t-1} + i_{k,t} * \tilde{c}_{k,t}, \\
o_{k,t} &= \sigma(W_{k,o} \cdot [h_{k,t-1}, z_{t-1}] + b_{k,o}), \\
h_{k,t} &= o_{k,t} * \tanh(c_{k,t}).
\end{aligned} \tag{4.28}$$

In the same way as the m-GRU, the m-LSTM has another LSTM, called $LSTM_c$, which will be used to encode the covariates that will be used to estimate $\vec{\pi}_t$. The third and final model proposed for our framework is the Temporal Kolmogorov Arnold Networks (TKAN), The TKAN have been introduced in the previous sections. Its excellent results shown in Chapter 3 have motivated us to propose a switching extension based on this new architecture. As we have done previously, we will use the notation m-TKAN to designate switching TKAN. Keeping the same

notation as the one used for the GRU and the LSTM, $\text{TKAN}_{k,in}$ would be given by,

$$\begin{aligned}
f_{k,t} &= \sigma(W_{k,f}x_t + U_{k,f}h_{k,t-1} + b_{k,f}),\\
i_{k,t} &= \sigma(W_{k,i}x_t + U_{ki}h_{t-1} + b_{k,i}),\\
r_{k,t} &= \text{Concat}[\phi_{k,1}(s_{k,1,t}), \phi_{k,2}(s_{k,2,t}), ..., \phi_{k,L}(s_{k,L,t})],\\
o_{k,t} &= \sigma(W_{k,o}r_{k,t} + b_{k,o}),\\
c_{k,t} &= f_{k,t} \odot c_{k,t-1} + i_{k,t} \odot \tilde{c}_{k,t},\\
h_{k,t} &= o_{k,t} \odot \tanh(c_{k,t}),
\end{aligned} \tag{4.29}$$

where $s_{k,l,t} = W_{k,l,\tilde{x}}x_t + W_{k,l,\tilde{h}}\tilde{h}_{k,l,t-1}$ is the input of each RKAN, $\tilde{c}_{k,t} = \sigma(W_c x_t + U_c h_{t-1} + b_c)$ represents its internal memory, and $\phi_{k,l}$ is a KAN layer of regime k. The "memory" step $\tilde{h}_{k,l,t}$ is defined as a combination of past hidden states for each k regime, such,

$$\tilde{h}_{k,l,t} = W_{hh}\tilde{h}_{k,l,t-1} + W_{k,hz}\tilde{o}_t, \tag{4.30}$$

Similar to the m-GRU and m-LSTM, the m-TKAN model has a TKAN layer called $TKAN_c$. This layer is responsible for encoding the covariates, an additional input variables denoted 2_t, to estimate the probabilities $\vec{\pi}_t$. After estimating the probabilities and deducing the predicted regimes, we backtest a simple strategy for each of the models. Depending on the prediction made for $\hat{s}_t$, we open a long or short position if the predicted regime is bullish or bearish. All the results of these backtests are available in Appendix 7.3-7.4-7.5.

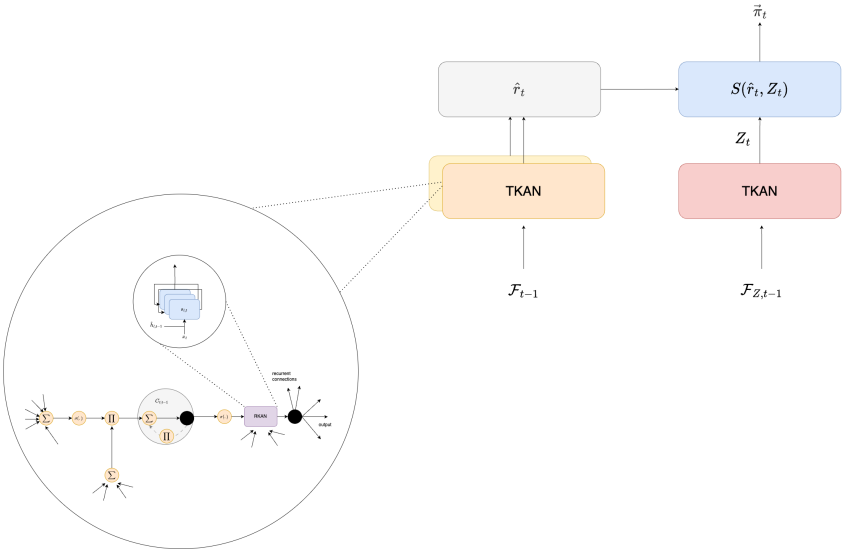


Figure 4.12 – Structure of the m-TKAN

The following section section will also examines the results obtains for these backtests.

4.3 Learning Task

To proceed the estimation of regime switching parameters, we download the open, high, low, close (OHLC) of Bitcoin from cryptocompare.com and we compute the log return defined as:

$$r_t = log(p_t) - log(p_{t-1}),$$

then we build our HML and IV (4.12). These covariates will be stacked in a vector with a fixed sequence length. The input vector $Z_{0:T-1}$ will be standardized and divided by the maximum of absolute value.

$$Z_{0:T-1} := \{\{HML_t\}_{t=0}^{T-1}, \{IV_t\}_{t=0}^{T-1}\}.$$

Creating a learning task to predict regime is not as straightforward as it is for many other models, as the real states are not known. In order to test wether our model is efficient in predicting, we had to labelize our datas with regime. To do so, we use a simple systematic methods that consists as defining two regimes, a bull one when the average price of the 20 past days (including the observation at t when we make our prediction) is lower than the average price of the 20 next days, a bear one in the opposite case. Having such, the tasks become a standard classifcation tasks, where we have to predict in which class we are, given the current information in t. The outputs of the model being two probability, we encode the classes using a one hot encoder, and thus calibrate the model using a categorical cross-entropy as loss, which is the standard for this kind of problem. The Categorical cross-entropy

$$L = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(p_{ic})$$

where N is the number of samples, y_{ic} is the binary indicator (0 or 1) indicating whether the class label c is the correct classification for sample i, and p_{ic} is the predicted probability that the sample i belongs to the class c. For each sample i and each class denoted c, the binary indicator y_{ic} is 1 if the sample belongs to class c and 0 otherwise. This loss function commonly used in classification tasks to measure the difference between two probability distributions: true labels and predicted labels. Finally, we used a validation set during the training, in order to use an early stopping callback that stop the training after 10 consecutive epochs without improvement of the validation loss, as well as a learning rate reduction by a factor of 4 after 5 consecutive epochs without improvments. These two together reduce the risks of overfitting and enables to have a systematic approach on this learning rate selection. We used RNNs as the neural network parts in our model, as we added a sequence dimension to the input, in order to be able to represent the markov-chain. We thus compared the two most standards RNN that are the GRU and LSTM, but also the TKAN. Finally, in order to test the different RNNs the same way, we build all the model the same, with 2 layers of 100 units in each, using their standard activation functions. Only the TKAN as a bit more hyper-parameter, with 3 internal RKAN

layers of degree 3 and grid-size 5 which are the defaults of the models, and an internal KAN sub-layers output dimension of 10.

During the training task, we know that neural networks can tend to adapt to the training data, and this is due to the large number of iterations, and then become unable to generalise what they have learned in the training phase to perform well on the test set. One way to overcome this problem is to track the evolution of the error on the training and validation sets at each iteration and analytically find out at which iteration the error on the test set increases while that on the training set continues to decrease. This technique allows us to select the parameters of our model without overfitting bias.

We do not seek to fine-tune our extended models but to assess the ability of our predictor to identify market regimes, and predict the next market regime. We also seek to stabilize the transition probabilities which seem to be very sensitive to the presence of some covariates during the estimation process for the switching Markov models. We believe that neural networks will help stabilize the probability of avoiding the transition from one state to another. Indeed, the complexity and the number of coefficients that we will have in the most complete models will reduce this increased sensitivity to covariates. For the learning task we have built a training set of 80% of the total observations we have, and among this 80% we use 20% of this training set to build a validation with an early stopping mechanism in order to avoid overfitting.

5 Results

Looking at the results on the test set, it appears that the m-GRU model has a high number of false positives compared to true negatives. This indicates it tends to incorrectly classify negatives as positives. However, the true positives are higher than false negatives. This suggests it performs better in correctly identifying positive cases. The m-LSTM model shows a better performance with fewer false positives and a higher number of true negatives compared to the GRU model. However, it has a slightly higher number of false negatives and fewer true positives. It indicates a bit of a trade-off in correctly identifying positive cases. The TKAN model shows the best performance in terms of minimizing false positives. TKAN obtained the highest number of true negatives among the three models. It also maintains a reasonable balance between false negatives and true positives, indicating a good overall performance in identifying both positive and negative cases accurately. Results obtained during the training task are available in the appendix.

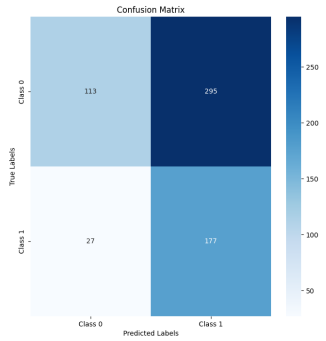


Figure 4.13 – GRU Confusion matrix (out of sample)

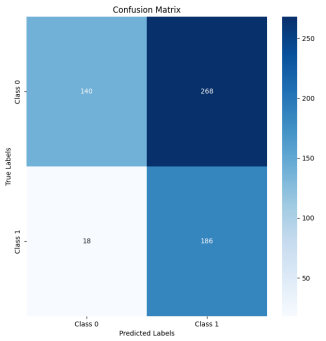


Figure 4.14 – LSTM Confusion matrix (out of sample)

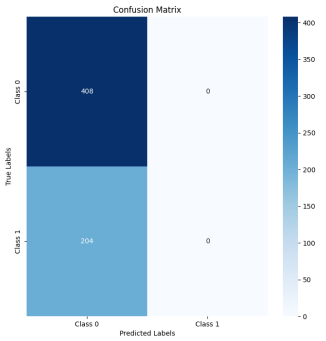


Figure 4.15 – TKAN Confusion matrix (out of sample)

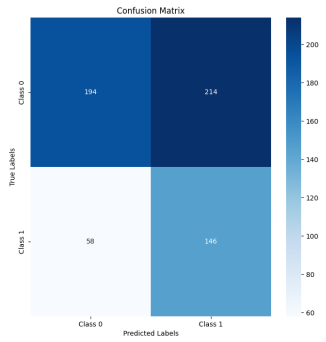


Figure 4.16 – m-GRU Confusion matrix (out of sample)

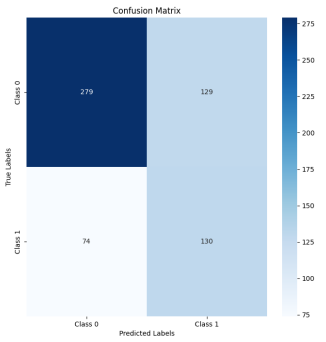


Figure 4.17 – m-LSTM Confusion matrix (out of sample)

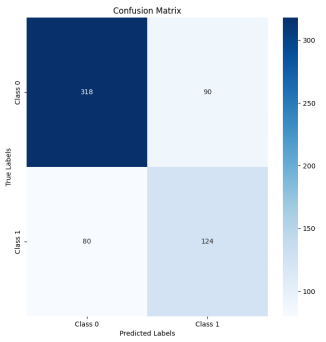


Figure 4.18 – m-TKAN Confusion matrix (out of sample)

Table 4.4 – Comparison of simple RNNs versus Switching RNNs

Model	**Class**	**Precision**	**Recall**	**F1-Score**	**Support**	**Accuracy**
LSTM vs. m-LSTM						
LSTM	Class 0	**0.89**	0.34	0.49	408	0.53
	Class 1	0.41	**0.91**	0.57	204	
m-LSTM	Class 0	0.79	**0.68**	**0.73**	408	**0.67**
	Class 1	**0.50**	0.64	**0.56**	204	
GRU vs. m-GRU						
GRU	Class 0	**0.81**	0.28	0.41	408	0.47
	Class 1	0.38	0.87	0.52	204	
m-GRU	Class 0	0.77	**0.48**	**0.59**	408	**0.56**
	Class 1	**0.41**	**0.72**	**0.52**	204	
TKAN vs. m-TKAN						
TKAN	Class 0	0.67	**1.00**	**0.80**	408	0.67
	Class 1	0.00	0.00	0.00	204	
m-TKAN	Class 0	**0.79**	0.78	0.79	408	**0.72**
	Class 1	**0.58**	**0.59**	**0.58**	204	

Table 4.4 shows that our Switching neural networks enhance the model's capacity to learn and predict meaningful regimes. Indeed, conventional RNNs are unable to identify market regime, as evidenced by the superior performance of TKAN, which merely identifies the dominant class. However, our Switching models demonstrate higher performance in forecasting for all models. Despite the quality of GRU-based models, those incorporating TKAN units exhibit notable accuracy. The TKAN-based model, in particular, demonstrates robust performance on external tasks.

	m-GRU	m-LSTM	m-TKAN
Mean Return (μ)	0.754659	0.794480	**1.131593**
Standard Deviation (σ)	0.734199	0.734084	0.732871
Sharpe Ratio	1.027867	1.082273	**1.544053**
Max Drawdown	-1.071958	-1.254106	-0.740906
Sortino Ratio	1.783035	1.863751	2.690941
Mean Daily Turnover	0.173849	0.109761	0.360958
Annual Turnover	63.454880	40.062615	131.749540
Mean Return on Volume	0.011893	0.019831	0.008589
Beta	-0.142187	-0.045676	0.154983
Alpha	0.880322	0.834848	0.994620

Table 4.5 – Performance table (In Sample)

Table 4.5 shows the results obtained on the training task (in sample estimation) using the m-GRU, m-LSTM and m-TKAN. Results show significant big differences in their predictive capabilities between train and test set. During the training phase, TKAN stands out for its high average return (1.131593) and a higher Sharpe (1.544053) and Sortino (2.690941) ratios. These ratios suggest a significantly better risk-adjusted performance than the other models. The m-LSTM, demonstrating a Sharpe ratio of (1.082273) and a Sortino ratio of (1.863751), also performed commendably, although it remains slightly inferior to the m-TKAN. Conversely, the m-GRU encountered more challenges in its performance. Despite a good average return (0.754659), it has a lower Sharpe ratio (1.027867) and Sortino ratio (1.783035), as well as a higher MDD (-1.071958).

	m-GRU	m-LSTM	m-TKAN
Mean Return (μ)	-0.398766	0.198593	**0.444902**
Standard Deviation (σ)	0.490487	0.490821	0.490378
Sharpe Ratio	-0.813001	0.404614	**0.907263**
Max Drawdown	-0.687949	-0.342666	-0.467687
Sortino Ratio	-1.219218	0.659887	1.497058
Mean Daily Turnover	0.173486	0.163666	0.468085
Annual Turnover	63.322422	59.738134	170.851064
Mean Return on Volume	-0.006297	0.003324	0.002604
Beta	-0.043611	0.156359	0.374576
Alpha	-0.361273	0.064168	0.122870

Table 4.6 – Performance table (Out of Sample)

The Table 4.6 reveals impressive metrics for the m-TKAN during testing (out of sample estimation). The m-LSTM also does very well on the test sample. The TKAN maintained good risk control, with a moderate max drawdown (-0.467687) and a high Sortino ratio (1.497058). The m-GRU, on the other hand, shows a negative test performance with an average return of -0.398766 and unfavorable risk ratios, underlining a poorer ability to generalize. The m-LSTM and m-TKAN appear to be more robust and efficient models for managing sequential data in a variety of market environments, with m-TKAN standing out in particular for its ability to maximize risk-adjusted returns.

6 Conclusion

In conclusion, we have seen that switching models are particularly beneficial for analyzing digital assets. Their relevance can be explained by the highly volatile dynamic nature of this new market, still in its beginning. These models are very effective at capturing rapid transitions between different states (bullish or bearish). They adapt quickly and efficiently to the influence of external factors such as regulatory or technological changes. They are able to track the structural evolution of this market. These models provide a robust analytical framework for

understanding the complex dynamics of the digital asset market. In this chapter, we proposed the incorporation of Markov switching into recurrent neural network models, an innovative framework that improves the performance of these models. Particularly in the case of TKAN, this new state-space framework allows for a substantial improvement in the ability to capture and predict significant regimes in sequential data. The m-TKAN shows the most significant improvement, evolving from a model unable to classify class 1 to a model performing well on both classes. In the context of financial data, this improvement translates into superior financial performance and better risk management. Looking at the other models, those incorporating the Markov switching framework (m-LSTM, m-GRU, m-TKAN) demonstrate better overall predictive capacity than their conventional counterparts. The Markov switching models tend to offer more balanced performance between classes, whereas the classical models tend to favor one class over another. This study underlines the effectiveness of integrating Markov chain structures into recurrent neural network models, enhancing their ability to process complex sequential data and identify different regimes or classes.

7 Appendix

7.1 Regime Switching without TVTP (3 states)

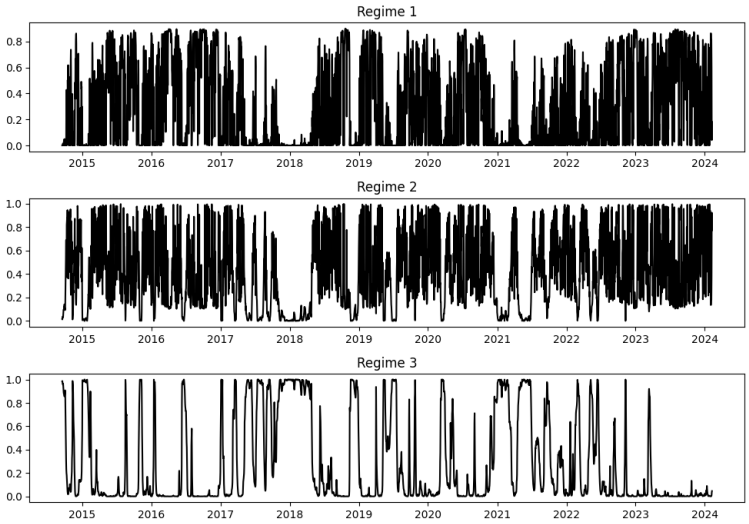


Figure 4.19 – Smoothed Marginal Probabilities

	coef	std err	z	P> |z|	[0.025	0.975]
const	0.0007	0.000	1.544	0.123	-0.000	0.002
sigma2	7.957e-05	2.22e-05	3.585	0.000	3.61e-05	0.000
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0028	0.001	2.433	0.015	0.001	0.005
sigma2	0.0010	0.000	4.466	0.000	0.001	0.001
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0003	0.002	0.137	0.891	-0.004	0.005
sigma2	0.0030	0.000	9.709	0.000	0.002	0.004
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
p[1->1]	0.6461	0.101	6.386	0.000	0.448	0.844
p[2->1]	0.2840	0.116	2.442	0.015	0.056	0.512
p[3->1]	4.88e-06	0.112	4.37e-05	1.000	-0.219	0.219
p[1->2]	0.3306	0.074	4.440	0.000	0.185	0.477
p[2->2]	0.6970	0.104	6.721	0.000	0.494	0.900
p[3->2]	0.0524	0.080	0.657	0.511	-0.104	0.209

7.2 Regime Switching TVTP (3 states)

Regime Switching HML, TVTP with HML Factor

	coef	std err	z	P> |z|	[0.025	0.975]
const	0.0003	0.002	0.165	0.869	-0.003	0.004
hml	-0.0010	0.003	-0.355	0.723	-0.007	0.005
sigma2	7.925e-05	4.09e-05	1.936	0.053	-9.65e-07	0.000
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0032	0.001	2.671	0.008	0.001	0.006
hml	-0.0002	0.004	-0.056	0.955	-0.008	0.008
sigma2	0.0010	0.001	1.725	0.085	-0.000	0.002
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	-0.0001	0.003	-0.041	0.967	-0.006	0.006
hml	-0.0001	0.003	-0.042	0.967	-0.006	0.006
sigma2	0.0032	0.000	9.058	0.000	0.002	0.004
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
p[1->1].const	2.1642	10.353	0.209	0.834	-18.127	22.455
p[2->1].const	2.5755	5.538	0.465	0.642	-8.279	13.431
p[3->1].const	-7.2483	3.618	-2.004	0.045	-14.339	-0.158
p[1->1].hml	-2.9672	4.644	-0.639	0.523	-12.069	6.135
p[2->1].hml	-0.0084	5.774	-0.001	0.999	-11.324	11.308
p[3->1].hml	0.3645	1.193	0.305	0.760	-1.974	2.703
p[1->2].const	2.6221	10.477	0.250	0.802	-17.913	23.157
p[2->2].const	3.3448	5.472	0.611	0.541	-7.381	14.070
p[3->2].const	-2.6787	0.581	-4.611	0.000	-3.817	-1.540
p[1->2].hml	-0.6451	4.562	-0.141	0.888	-9.587	8.297
p[2->2].hml	-0.0233	5.892	-0.004	0.997	-11.572	11.526
p[3->2].hml	0.1108	1.370	0.081	0.936	-2.574	2.795

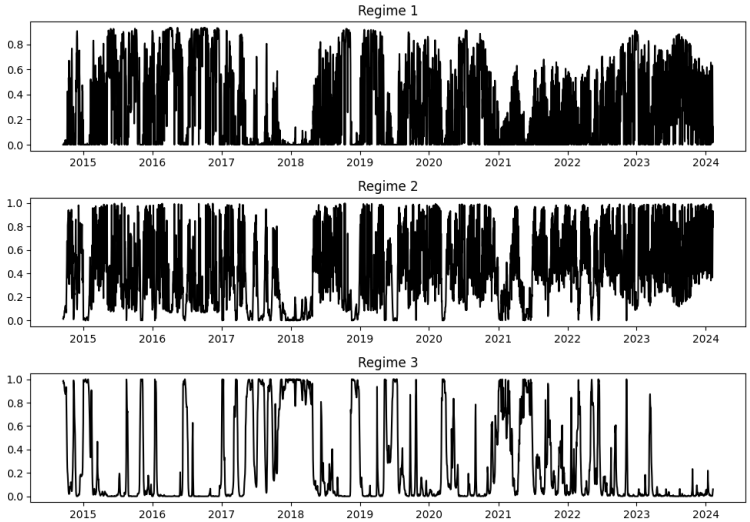


Figure 4.20 – Smoothed Marginal Probabilities

Regime Switching TVTP with HML Factor

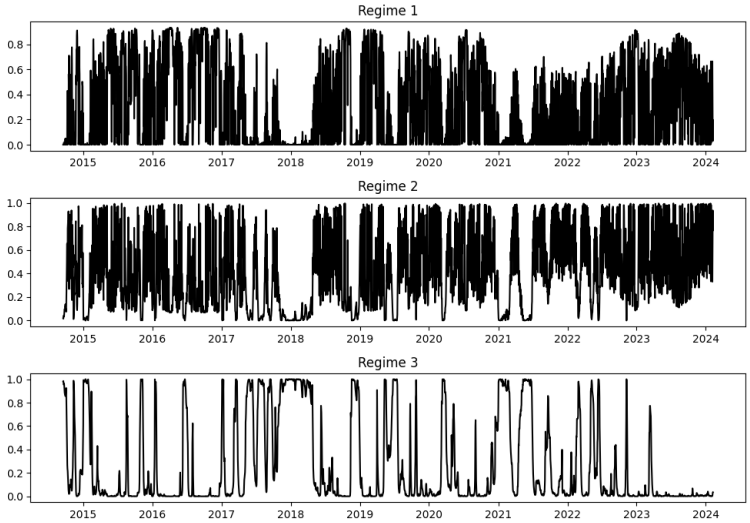


Figure 4.21 – Smoothed Marginal Probabilities

	coef	std err	z	P> |z|	[0.025	0.975]
const	0.0008	0.000	1.764	0.078	-8.87e-05	0.002
sigma2	8.456e-05	2.23e-05	3.784	0.000	4.08e-05	0.000
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0030	0.001	2.454	0.014	0.001	0.005
sigma2	0.0010	0.000	5.058	0.000	0.001	0.001
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	-0.0002	0.002	-0.072	0.943	-0.005	0.005
sigma2	0.0031	0.000	10.768	0.000	0.003	0.004
	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
p[1->1].const	2.3747	1.593	1.491	0.136	-0.747	5.496
p[2->1].const	3.3433	1.706	1.960	0.050	-0.000	6.687
p[3->1].const	-7.5457	4.747	-1.590	0.112	-16.849	1.758
p[1->1].hml	-2.1235	2.758	-0.770	0.441	-7.528	3.282
p[2->1].hml	1.0712	3.675	0.291	0.771	-6.132	8.275
p[3->1].hml	0.6017	0.257	2.341	0.019	0.098	1.105
p[1->2].const	2.6377	1.496	1.763	0.078	-0.295	5.571
p[2->2].const	4.0863	1.534	2.664	0.008	1.080	7.093
p[3->2].const	-2.8440	0.417	-6.816	0.000	-3.662	-2.026
p[1->2].hml	-0.0310	2.791	-0.011	0.991	-5.502	5.440
p[2->2].hml	1.1641	3.641	0.320	0.749	-5.972	8.300
p[3->2].hml	-0.3378	0.369	-0.915	0.360	-1.061	0.386

Table 4.7 – Model Parameters

7.3 Switching GRU

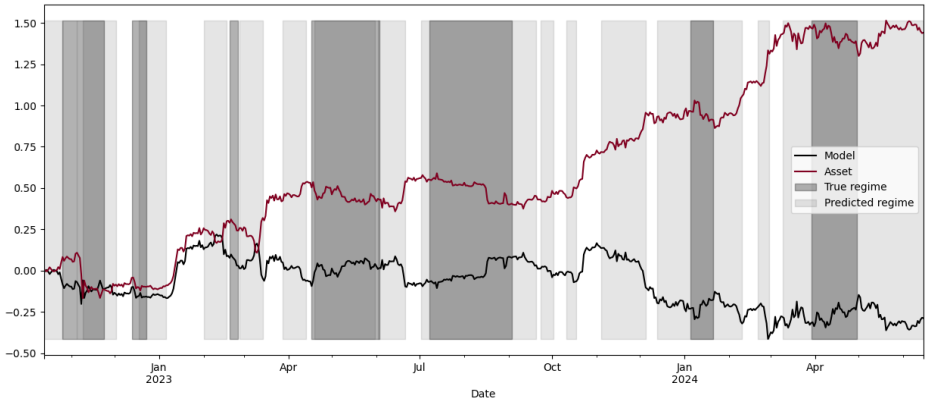


Figure 4.22 – GRU (Out of Sample)

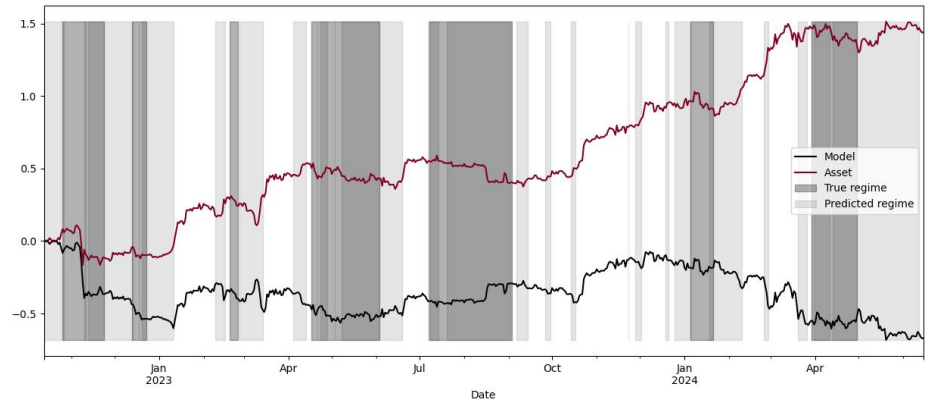


Figure 4.23 – m-GRU (Out of sample)

7.4 Switching LSTM

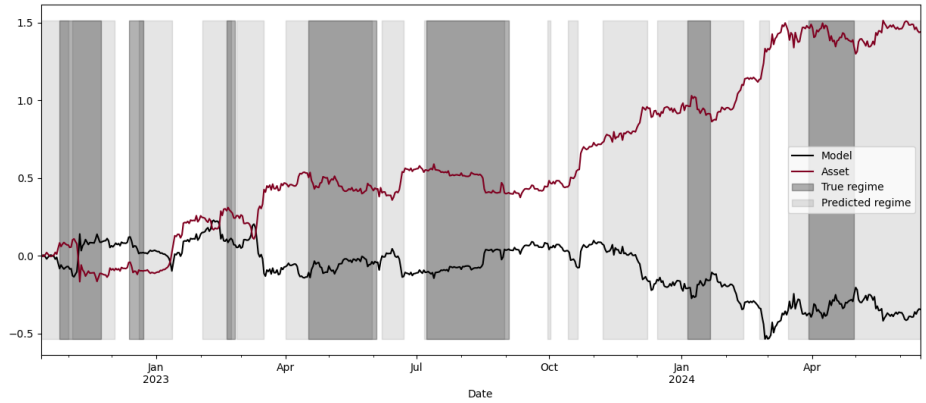


Figure 4.24 – LSTM (Out of sample)

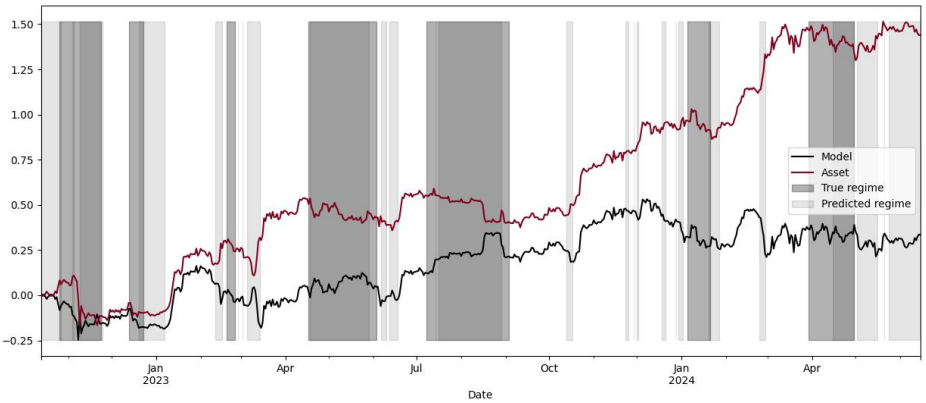


Figure 4.25 – m-LSTM (Out of sample)

7.5 Switching TKAN

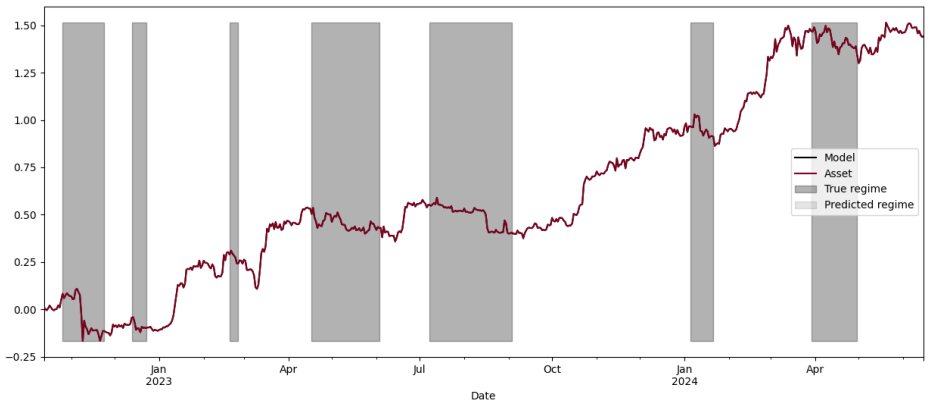


Figure 4.26 – TKAN (Out of sample)

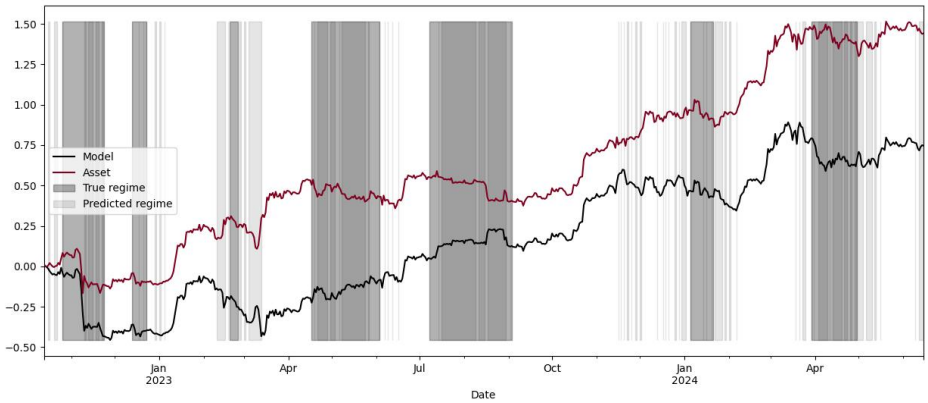


Figure 4.27 – m-TKAN (Out of sample)

Chapter 5

A New View on Markov-Switching GARCH Models

This part is a joint work with Jean-David Fermanian (CREST, ENSAE).

Abstract

The popular Markov-Switching GARCH model of [HMP04] suffers from a lack of financial intuition. Here, it is revisited and slightly modified to obtain more intuitive specifications. The price to be paid with our new models comes from path dependencies for inference purpose by maximum likelihood. A strategy based on Nonparametric Simulated Maximum Likelihood is proposed and its performances are evaluatd by simulation.

Contents

1 Introduction

Since the introduction of Markov-Switching models for the econometric analysis of economic cycles by Hamilton [Ham89a], this family of models has been extensively studied in the academic literature and among practitioners. They are based on the intuitive idea that there exist different "states of the world / economy / financial markets" that induce different dynamics for economic and / or financial variables. Traditionally, the dynamics of the latter states is random and driven by a discrete Markov chain that is not observable (also said latent or hidden, equivalently). Dealing with some parametric models means that these parameters are time-dependent, and their current value at any time t depends on the state of the world s_t at that date.

In other words, a Markov-Switching parametric model for some time series $(X_t)_{t\in\mathbb{Z}}$ is described by a family of distributions $\{P_\theta, \theta \in \Theta\}$, and by a Markov chain (s_t) that can take m values (the so-called "Data generating process"). When the Markov chain is homogeneous (the typical situation), the law of this Markov chain is deduced from a $m \times m$ transition matrix M, so that

$$M = [p_{ij}],\ p_{i,j} := \mathbb{P}(s_t = j \,|\, s_{t-1} = i),\ (i,j) \in \{1,\dots,m\}^2.$$

Thus, the law of X_t given the available information $\mathcal{F}_{t-1}$ until (and including) $t-1$ will be denoted P_{θ_t}, where the current parameter θ_t will depend on s_t and is measurable w.r.t. $\mathcal{F}_{t-1}$. The latter sigma-algebra contains the past values $X_{t-1}, X_{t-2}, \dots$ plus, possibly, some additional exogenous information (that were available until and just before t). In such models, the dynamics of the successive states (s_t) most often does not depend on the innovations of the process (X_t), when the conditional law of X_t depends on s_t. For example, assume two underlying states $\{1,2\}$ in an economy (i.e. $m = 2$) called "growth" and "recession". The dynamics of a vector of macro-economic variables X_t may be set as the vectorial auto-regressive switching regime model

$$X_t = A_{s_t} + B_{s_t} X_{t-1} + \epsilon_t,\ \epsilon_t \sim \mathcal{N}(0, \Sigma_{s_t})$$

for some vectors (resp. matrices) A_1 and A_2 (resp. B_1 and B_2) and some covariance matrices Σ_1 and Σ_2.

Markov switching regime models have to be distinguished from mixture models. In the latter case, the law of X_t given $\mathcal{F}_{t-1}$ is randomly drawn among a set of candidates, but independently from one date to the next. Formally, this family of models can be seen as a particular case of Markov-Switching models with a very special transition matrix M for which $\mathbb{P}(s_t = j \,|\, s_{t-1} = i) = \mathbb{P}(s_t = j)$ for every (i,j). In other words, the latter Markov chain has no memory. Since this feature has strong consequences in terms of probabilistic and statistical properties, the two classes of models are distinguished in the literature. See [Frü06] for a very complete overview of these two competing perspectives.

The natural idea of applying switching regime models in financial econometrics rapidly appeared but was not so straightforward. Indeed, asset return distributions exhibit well-known and peculiar empirical features: fat tails (extreme events are significantly more frequent than with Gaussian distributions), skewness (due to the "leverage effect", the probability of observing large

negative returns is typically more frequent that for positive returns), and particularly volatility clustering (some periods of high volatility are followed by some periods of low volatility, due to some "herding behaviors" of dealers in financial markets). Since the latter feature is traditionnally captured by GARCH-type models, some authors have proposed Markov-Switching GARCH (MS-GARCH, to be short) models to combine the idea of different regimes/cycles and the persistence of return volatilities. The most natural model specification is then as follows: assume that some series of financial return $(r_t)_{t=1,\dots,T}$ follows the dynamics

$$r_t = \mu_{s_t} + \sigma_t \epsilon_t, \tag{5.1}$$

where

— the innovations/noises (ϵ_t) is a sequence of independent and identically distributed random variables (or martingale differences more generally). Their means are zero and their variances are equal to one.

— (s_t) follows a discrete Markov chain and this process is independent of the noises (ϵ_t). We assume there will be m underlying states, denoted $1, 2, \dots, m$.

— there are m values $\mu_1, \dots, \mu_m$ for the conditional means.

— the process of conditional volatilities satisfies

$$\sigma_t^2 = \omega_k + a_k z_{t-1}^2 + b_k \sigma_{t-1}^2, \text{ when } s_t = k,\ k \in \{1, \dots, m\}. \tag{5.2}$$

and z_t denotes the "unexpected" change in the asset return at time t. Typically, depending on the authors, it is assumed that $z_t = r_t - \mu_{s_t}$, $z_t = r_t - \mathbb{E}_{t-1}[\mu_{s_t}]$, or even $z_t = r_t$ simply.

Obviously, (5.2) refers to the "usual" GARCH(1,1) model, but other more sophisticated and richer dynamics can be introduced instead: introduction of asymmetries, thresholds, more numerous lags, etc. Unfortunately, with all such specifications, the conditional variance σ_t^2 depends not just on the current regime s_t but on the entire past history of the process until $t-1$, inducing complexities in terms of inference and forecasting. In particular, the calculation of the likelihood becomes numerically untractable. Indeed, the conditional density of any observation r_t given the past (i.e. given $\mathcal{F}_{t-1}$) depends on all (unobservable) $\sigma_{t'}$, $t' \leq t$ and then on all past observations in a complex way. The first attempts to solve such "path dependencies" were due to [Gra96] and [Kla02], but at the price of questionable and rather ad-hoc modifications of Equation (5.2). See Appendix 7.2 for details.

A more convenient and sounder proposition has been made by [HMP04]. They assumed the existence of m "parallel" dynamics

$$\sigma_{k,t}^2 = \omega_k + a_k r_{t-1}^2 + b_k \sigma_{k,t-1}^2, \text{ when } k \in \{1, \dots, m\}. \tag{5.3}$$

when the asset return process is $r_t = \sigma_{s_t,t}\epsilon_t$. Note that the conditional expectation of r_t is assumed to be zero here (i.e. no μ_{s_t}). Thus, it can be checked that the latter process does no longer suffer from path dependencies for inference purpose. In particular, the estimation of the

model given by (5.3) and in [HMP04] can be led by maximum likelihood thanks to the use of the Hamilton filter (as for "usual" Markov-Switching regression models, in the spirit of [Ham89a]). Alternative methods are available, notably through Gibbs sampling under a Bayesian perspective ([BPR10]), by a mix of EM-algorithm and importance sampling ([Aug14]), or by the General Method of Moments ([FZ+08]). A R-package is available for estimating this family of models, with some variants of (5.3) and several potential laws for ϵ_t.

In particular, the dynamics of crypto-asset returns exhibit special features that cannot be easily captured with simple dynamic models. Many authors suggest the existence of several regimes in such markets, notably three regimes: "boom", "bust" and "nothing special". This has fuelled a burgeoning empirical literature that rapidly expands. In particular, some authors have logically tested MS-GARCH models as discussed above: see [Ard+18; ABR19; CZ19; PPS22], among others. Most papers insist on the relevance of considering several regimes to describe volatility dynamics.

Almost all authors in the literature about MS-GARCH models assume that the conditional means μ_{s_t} are zero, mainly for convenience or to avoid the introduction of path dependencies in their model specifications. However, it is relatively easy to modify the methodology of [HMP04] by assuming $r_t = \mu_{s_t} + \sigma_{s_t,t}\epsilon_t$ and

$$\sigma^2_{k,t} = \omega_k + a_k(r_{t-1} - \mu_k)^2 + b_k\sigma^2_{k,t-1}, \text{ for any } k \in \{1, \ldots, m\}. \tag{5.4}$$

This extension of the core model of [HMP04] had been mentioned by these authors in passing (Section 2.3.2 in [HMP04]). Since the crypto assets seem to exhibit momentums, the possibility of non-zero conditional should be particularly relevant for such assets. Since the inference of Model (5.4) has not been specified in the literature to the best of our knowledge, an adapted maximum likelihood method is detailed in Section 7.1 in the appendix.

2 MS-Garch model of Haas et al. (2002) revisited

Coming back to Equation (5.3), it appears that Haas et al. (2004)'s model, even convenient in practice (for inference purpose particularly), suffers from some weaknesses in terms of interpretability and does not support well financial intuition. Indeed, an observation, say at $t-1$, associated with a particular state at that time will update the conditional volatility $s_{s_t,t}$ at time t, whatever the value of s_t. Thus, for any $j \in \{1, \ldots, m\}$, the process $(\sigma_{j,t})_{t\in\mathbb{Z}}$ will be "contaminated" by the observations associated with all the other past states.

To illustrate, imagine two states and two different volatility dynamics: $j = 1$ is related to a "quiet" regime and $j = 2$ to an "agitated market", typically. On average, $\sigma_{1,t}$ will be smaller than the $\sigma_{2,t}$. In other words, $\mathbb{E}[|r_t - \mu_1| \,|\, s_t = 1]$ will be a lot smaller than $\mathbb{E}[|r_t - \mu_2| \,|\, s_t = 2]$. Assume that $s_t = 2$ but that the past states were $s_{t-k} = 1$, $k > 0$, during a rather long period of time (i.e. the first state is persistent); then, the average value of $\sigma^2_{2,t}$ will be close to $\omega_2/(1-\beta_2)$. If we had followed the intial intuition of [HMP04] about the interpretation of different hidden volatility processes, we rather expect an average value close to $\omega_2/(1-a_2-\beta_2)$. As a consequence,

if state 2 is not very persistent, the time spent under $s_t = 2$ will most often not be long enough for $\sigma^2_{2,t}$ to reach and turn around its "natural target" $\omega_2/(1-a_2-\beta_2)$. This conditional variance will be most of the time "biased downwards", keeping in mind the interpretation in terms of several parallel dynamics.

To summarize, it is difficult to have a clear meaning of the processes $(\sigma_{j,t})$, $j \in \{1,\dots,m\}$ under the specification (5.3) or (5.4). To be more in line with the idea of several parallel volatility dynamics, we now propose to modify the latter equations.

Note that (5.4) can be rewritten

$$\sigma^2_{k,t} = \omega_k + a_k \sigma^2_{s_{t-1},t-1}\epsilon^2_{t-1} + b_k \sigma^2_{k,t-1}, \text{ for any } k \in \{1,\dots,m\}. \tag{5.5}$$

The first new model would be now

$$r_t = \mu_{s_t} + \sigma_{s_t,t}\epsilon_t, \text{ and} \tag{5.6}$$

$$\sigma^2_{k,t} = \omega_k + a_k \sigma^2_{k,t-1}\epsilon^2_{t-1} + b_k \sigma^2_{k,t-1}, \text{ for any } k \in \{1,\dots,m\}. \tag{5.7}$$

In the latter equations, the innovations (ϵ_t) themselves but not the observations are used to update the conditional volatilities. Obviously, since innovations are not directly observable, inference under this new approach will be more complex than for (5.5) in practice even if our new model appears to us as fundamentally more satisfying.

In (5.7), the same innovations are used to update all volatility processes. Still under the point of view of parallel dynamics, it can be argued that different innovation processes could be considered for different underlying regimes. Therefore, for some agitated states, this would allow us to introduce fat-tailed and/or asymmetric innovations by conveniently choosing their distributions, when a standard $\mathcal{N}(0,1)$ should be sufficient to model the law of innovations in quiet/standard states.

This motivates us to introduce another new model:

$$r_t = \mu_{s_t} + \sigma_{s_t,t}\epsilon_{s_t,t}, \tag{5.8}$$

with the updating volatility equations become

$$\sigma^2_{k,t} = \omega_k + a_k \sigma^2_{k,t-1}\epsilon^2_{k,t-1} + b_k \sigma^2_{k,t-1}, \text{ for any } k \in \{1,\dots,m\}. \tag{5.9}$$

Here, we have introduced an iid sequence of (column) vectors of innovations $\vec{\epsilon}_t := (\epsilon_{1,t},\dots,\epsilon_{m,t})$ at every date. Its law is m-dimensional and has to be fixed, under the constraint that $\mathbb{E}[\vec{\epsilon}_t] = 0$ and $\mathbb{E}[\epsilon^2_{j,t}] = 1$ for every j and t. In particular, it is not mandatory to assume that the components of $\vec{\epsilon}_t$ are mutually independent, even if such an assumption (called MIA hereafter) is rather natural. Obviously, the vectors of innovations are not observed. At time t, the law of r_t given the current state $s_t = k$ depends on $\epsilon_{k,t}, \epsilon_{k,t-1},\dots$ but not on the other innovations under MIA. Thus, there will be numerous hidden quantities compared to (5.3) or even (5.7). Inference

of (5.8)-(5.9) will then be challenging due to the potentially large number of latent variables.

Under (5.9) and (MIA), the new model (5.9) has built m independent univariate GARCH(1,1) processes that are only partially observed. In terms of theoretical properties, this can be seen as a very peculiar multivariate GARCH process: see Section 11.2 in [FZ19] and the survey [BLR06], for instance. The challenge will be to estimate these m dynamics knowing we only observe the series (y_t). This will be the purpose of Section 4. We have assumed that the processes are indexed for any $t \in \mathbb{Z}$, and then start "from infinity". This simplifies the statement of stationarity properties.

The discrete Markov chain $(s_t)_{t\in\mathbb{Z}}$ on $\{1,\dots,m\}$ is assumed to be homogeneous, ergodic and it admits an invariant distribution π s.t. $\pi M = \pi$, recalling its transition matrix M. When the process of asset returns starts at a fixed date, say t_0, we assume that the initial state s_{t_0} is drawn following the stationary law π.

It is not surprising that the processes (5.6)-(5.7) admit a unique strongly stationary when the m latent (usual) GARCH(1,1) processes share this property. Indeed, this result will be a consequence of the independence between the state process (s_t) and the innovation process (ϵ_t).

Proposition 2.1. *If, for any $k \in \{1,\dots,m\}$,*

$$-\infty \leq \gamma_k := \mathbb{E}\big[\ln(a_k\epsilon_t^2 + b_k)\big] < 0, \tag{5.10}$$

then there exists a unique strongly stationary solution $(r_t)_{t\in\mathbb{Z}}$ of Model (5.6)-(5.7). This solution is finite almost surely, nonanticipative and ergodic.

Proof. To prove the latter result, we rely on the strict stationarity of usual GARCH(1,1) processes. Define the maps $f_k : \mathbb{R} \to \mathbb{R}^+$, where $f_k(z) = a_k z^2 + b_k$, $k \in \{1,\dots,m\}$. By definition of our model, for every index k and every time t,

$$\sigma_{k,t}^2 = \omega_k + f_k(\epsilon_{t-1})\sigma_{k,t-1}^2.$$

As in the proof of Theorem 2.1 in [FZ19], deduce

$$\sigma_{k,t}^2 = \omega_k\Big(1 + \sum_{i\geq 1}\prod_{j=1}^{i} f_k(\epsilon_{t-j})\Big), \tag{5.11}$$

and this series is convergent a.s. by the Cauchy rule for series with nonnegative terms. Moreover, we can assume the successive states are drawn following the stationary law of their corresponding Markov chain. Therefore, setting

$$r_t = \Big\{\sum_{k=1}^{m} \mathbf{1}(s_t = k)\sigma_{k,t}\Big\}\epsilon_t = \Big\{\sum_{k=1}^{m} \mathbf{1}(s_t = k)\sqrt{\omega_k}\big(1 + \sum_{i\geq 1}\prod_{j=1}^{i} f_k(\epsilon_{t-j})\big)^{1/2}\Big\}\epsilon_t, \tag{5.12}$$

we get a solution of model (5.6)-(5.7). Thus, for any t, the latter solution r_t is a measurable and deterministic map of the process $(\epsilon_t, s_t)_{t\in\mathbb{Z}} = (\epsilon_t)_{t\in\mathbb{Z}} \otimes (s_t)_{t\in\mathbb{Z}}$. Since the latter one is stationary,

ergodic, this is the case of the process (r_t) itself (c.f. Theorem A.1 in [FZ19]). Moreover, since any r_t is a measurable map of the current and past innovations (and of the current states), the solution (5.12) is nonanticipative. To prove the uniqueness of (r_t), note that any solution of model (5.6)-(5.7) has to satisfy the relationship $r_t = \{\sum_{k=1}^m \mathbf{1}(s_t = k)\sigma_{k,t}\}\epsilon_t$. Under our assumptions, the processes $(\sigma_{k,t})$ are uniquely defined, proving the uniqueness of (r_t). □

Similarly, we prove the stationarity of the processes (5.8)-(5.9).

Proposition 2.2. *If, for any $k \in \{1, \dots, m\}$,*

$$-\infty \leq \tilde{\gamma}_k := \mathbb{E}\big[\ln(a_k\epsilon_{k,t}^2 + b_k)\big] < 0,$$

then there exists a unique strongly stationary solution $(r_t)_{t\in\mathbb{Z}}$ of model (5.8)-(5.9). This solution is finite almost surely, nonanticipative and ergodic.

Proof. This arguments are very close to those in the proof of Proposition 2.1. With the same notations, for every index k and every time t, we have

$$\sigma_{k,t}^2 = \omega_k + f_k(\epsilon_{k,t-1})\sigma_{k,t-1}^2.$$

As in the proof of Theorem 2.1 in [FZ19], deduce

$$\sigma_{k,t}^2 = \omega_k\Big(1 + \sum_{i\geq 1}\prod_{j=1}^{i} f_k(\epsilon_{k,t-j})\Big),\ k \in \{1, \dots, m\}, \tag{5.13}$$

and the latter series are convergent a.s. Assume the successive states are drawn following the stationary law of their Markov chain. Therefore, setting

$$r_t = \sum_{k=1}^m \mathbf{1}(s_t = k)\sigma_{k,t}\epsilon_{k,t} = \sum_{k=1}^m \mathbf{1}(s_t = k)\sqrt{\omega_k}\big(1 + \sum_{i\geq 1}\prod_{j=1}^{i} f_k(\epsilon_{k,t-j})\big)^{1/2}\epsilon_{k,t}, \tag{5.14}$$

we get a solution of model (5.8)-(5.9). Thus, for any t, the latter solution r_t is a measurable deterministic map of the process $(\epsilon_{1,t}, \dots, \epsilon_{m,t}, s_t)_{t\in\mathbb{Z}}$. Since the latter process is stationary and ergodic, this is the case of the process (r_t) itself (c.f. Theorem A.1 in [FZ19]). Moreover, since any r_t is a measurable map of current and past innovations, the solution (5.14) is nonanticipative. The proof of the uniqueness of (r_t) is similar to the above. □

Note that the unique strictly stationary solutions of the latter two models are explicitly written in Equation (5.12) and (5.14).

Remark 1. *Condition (5.10) is satisfied when $a_k + b_k < 1$. Indeed, Jensen's inequality yields*

$$\mathbb{E}\big[\ln(a_k\epsilon_t^2 + b_k)\big] \leq \ln\left(\mathbb{E}[a_k\epsilon_t^2 + b_k]\right) = \ln\left(a_k + b_k\right) < 0.$$

Remark 2. *When the process (r_t) starts from a fixed initial date, say $t = 0$, it is not strictly stationary in general, in the case of the two latter models. This property is true only when σ_0*

are drawn under its corresponding stationary distribution given by (5.11) or (5.13).

Remark 3. *Assume there exist some indices $k \in I \subset \{1, \dots, m\}$ for which $\tilde{\gamma}_k \geq 0$ and $\omega_k > 0$. Morover, assume $\mathbb{P}(s_{k,t} = 1) > 0$ for at least one index $k \in I$. What would be the behavior of (r_t) under (5.6)-(5.7) ? When some γ_k is strictly positive, then the series $\sum_{i\geq 1} \prod_{j=1}^{i} f_k(\epsilon_{t-j})$ tends to $+\infty$ a.s., due to the Cauchy rule. Since $\omega_k > 0$, the sequence $(\sigma_{k,t})$ tends to $+\infty$ a.s. too, recalling to (5.11). When some γ_k is zero, then the series $\sum_{i\geq 1} \prod_{j=1}^{i} f_k(\epsilon_{t-j})$ still diverge towards the infinity a.s. Indeed, otherwise, $\prod_{j=1}^{n} a_k(\epsilon_{t-j})$ tends to zero and then $\sum_{j=1}^{n} \ln\left(f_k(\epsilon_{t-j})\right) \to -\infty$ when $n \to \infty$, for some non zero probability. Nonetheless, due to the Chung-Fuchs thoeorem ([CT12], Section 5.4),*

$$\limsup_n \sum_{j=1}^{n} \ln\left(f_k(\epsilon_{t-j})\right) = +\infty, \text{ and } \liminf_n \sum_{j=1}^{n} \ln\left(f_k(\epsilon_{t-j})\right) = -\infty,$$

almost surely. This contradicts the convergence of $\sum_{i\geq 1} \prod_{j=1}^{i} f_k(\epsilon_{t-j})$. Therefore, for any $k \in I$, $\sum_{i\geq 1} \prod_{j=1}^{n} f_k(\epsilon_{t-j})$ tends $+\infty$ with n, and $\sigma_{k,t} = +\infty$ a.s. For any date t, the return r_t is then equal to the realisation of a stationary process if $s_{k,t} \notin I$, or r_t equals $\pm\infty$ (its sign is the same as the sign of $\epsilon_{k,t}$), when $s_{k,t} \in I$. The latter event occurs with a nonzero probability at any date, due to the independence between the state process and the process of innovations. The solution of our models (5.6)-(5.7) can then be considered as degenerate (non finite), even if it is formally strongly stationary. The same phenomenon obviously occurs with Model (5.8)-(5.9).

Let us state the second-order stationary properties of the latter processes.

Proposition 2.3. *Assume ω_k and $\mathbb{P}(s_t = k)$ are positive for any $k \in \{1, \dots, m\}$.*

(i) If there exists a second-order stationary and nonanticipative solution of Model (5.6)-(5.7) (resp. Model (5.8)-(5.9)), then $a_k + b_k < 1$ for every k.

(ii) Conversely, if $a_k + b_k < 1$ for every k, then the process $(r_t)_{t\in\mathbb{Z}}$ given by (5.12) (resp. (5.14)) and that is the unique solution of Model (5.6)-(5.7) (resp. Model (5.8)-(5.9)) is a weak white noise: for any t, the covariance $\mathbb{E}[r_t r_{t+p}]$ is zero when $p > 0$ and

$$\mathbb{E}[r_t^2] = \sum_{k=1}^{m} \mathbb{P}(s_t = k)\Big\{\frac{\omega_k}{1 - a_k - b_k} + \mu_k^2\Big\}.$$

Moreover, there exists no other second-order stationary and nonanticipative solution.

Proof. The proof is provided only for Model (5.6)-(5.7) because the arguments are the same for Model (5.8)-(5.9).

Let $(r_t)_{t\in\mathbb{Z}}$ be a weakly stationary solution. Denoting $p_k := \mathbb{P}(s_t = k)$, simple calculations

yield

$$\begin{aligned}\mathbb{E}[r_t^2] &= \sum_{k=1}^{m} p_k \mathbb{E}[r_t^2|s_t=k] = \sum_{k=1}^{m} p_k \mathbb{E}[(\mu_k+\sigma_{k,t}\epsilon_t)^2] \\ &= \sum_{k=1}^{m} p_k (\mathbb{E}[\sigma_{k,t}^2]\mathbb{E}[\epsilon_t^2] + \mu_k^2 + 2\mu_k \mathbb{E}[\sigma_{k,t}]\mathbb{E}[\epsilon_t]) \\ &= \sum_{k=1}^{m} p_k (\mathbb{E}[\sigma_{k,t}^2] + \mu_k^2), \end{aligned} \tag{5.15}$$

because $\sigma_{k,t}$ and ϵ_t are independent by assumption ((r_t) is a nonanticipative solution, which implies that every $\sigma_{k,t}$ depends on past innovations only). Since the solution (r_t) is weakly stationary, $E[r_t^2]$ does not depend on t and is a constant. Equivalently and setting $\nu_{k,t} := \mathbb{E}[\sigma_{k,t}^2]$, this means $\sum_{k=1}^m p_k \nu_{k,t}$ is a constant sequence whose value is denoted by C. Assume $a_1 + b_1 > 1$. By invoking (5.7) several times, we get

$$C > p_1(a_1+b_1)\nu_{1,t} > p_1(a_1+b_1)^2\nu_{1,t-1} > \ldots > p_1(a_1+b_1)^N \nu_{1,t-N},$$

for every integer N. Deduce $(\nu_{1,t})_{t\in\mathbb{Z}}$ tends to zero when $t \to -\infty$. Nonetheless, for any t, $\sigma_{1,t}^2 \geq \omega_1$ and then $\nu_{1,t} \geq \omega_1 > 0$, providing a contradiction. Thus, $a_1 + b_1 \leq 1$. Moreover, $a_1 + b_1 \neq 1$. Otherwise, Equation (5.7) would yield $\nu_{1,t} = \omega_1 + \nu_{1,t-1}$ for every $t \in \mathbb{Z}$. The latter relationship cannot be satisfied when $\omega_1 > 0$ because it would induce $\nu_{1,t} \to \infty$ when $t \to +\infty$. Since $\nu_{k,t} \geq 0$ for every (k,t), this implies $C = +\infty$, and then a contradiction. To summarize, we have obtained $a_1 + b_1 < 1$. Since the latter reasoning can be led for any index, we have proven that, for every k, $a_k + b_k < 1$.

Conversely, assume $a_k + b_k < 1$ for very k. For any $t \in \mathbb{Z}$, denote by $\mathcal{F}_t$ the σ-algebra induced by the observations $\{r_{t-i}, i \geq 0\}$. If $p > 0$, then

$$\mathbb{E}[r_t r_{t+p}] = \mathbb{E}[\sigma_{s_t,t}\sigma_{s_{t+p},t+p}\epsilon_t\epsilon_{t+p}] = \mathbb{E}\big[\sigma_{s_t,t}\sigma_{s_{t+p},t+p}\epsilon_t\mathbb{E}[\epsilon_{t+p}|\mathcal{F}_{t+p-1}]\big] = 0,$$

since $\sigma_{s_t,t}$, $\sigma_{s_{t+p},t+p}$ and ϵ_t are measurable w.r.t. $\mathcal{F}_{t+p-1}$ (remind the expansions (5.11) and (5.12)). Moreover, due to the strict (and then weak) stationarity of the processes $(\sigma_{k,t})_{t\in\mathbb{Z}}$, we have $\mathbb{E}[\sigma_{k,t}^2] = \omega_k + (a_k+b_k)\mathbb{E}[\sigma_{k,t}^2]$ due to (5.7). Recalling (5.24), this yields

$$\mathbb{E}[r_t^2] = \sum_{k=1}^{m} \mathbb{P}(s_t=k)\{\mathbb{E}[\sigma_{k,t}^2]\mathbb{E}[\epsilon_t^2] + \mu_k^2\} = \sum_{k=1}^{m} \mathbb{P}(s_t=k)\{\mu_k^2 + \frac{\omega_k}{1-a_k-b_k}\},$$

as announced.

Concerning the uniqueness of the solution, assume there exists another solution $(\tilde{r}_t)_{t\in\mathbb{Z}}$ of Model (5.6)-(5.7). Denote $\tilde{\sigma}_{k,t}$ its corresponding instantaneous volatilities. As for the processes $(\omega_{k,t})$, they satisfy

$$\tilde{\sigma}_{k,t}^2 = \omega_k + f_k(\eta_{t-1})\tilde{\sigma}_{k,t-1}^2,\ t \in \mathbb{Z}, k \in \{1,\ldots,m\}.$$

Denote $\tilde{\sigma}_{k,t}^2 - \sigma_{k,t}^2 = \Delta_{k,t}$. Deduce, for any k, t, q,

$$\Delta_{k,t} = f_k(\eta_{t-1})\Delta_{k,t-1} = \ldots = \prod_{j=1}^{q} f_k(\eta_{t-j})\Delta_{k,t-q}.$$

Since the solutions are nonanticipative and the map $a(\cdot)$ is nonnegative, we have

$$\mathbb{E}[|\Delta_{k,t}|] = \mathbb{E}[\prod_{j=1}^{q} f_k(\eta_{t-j})]\mathbb{E}[\Delta_{k,t-q}] = (a_k + b_k)^q \mathbb{E}[|\Delta_{k,t-q}|].$$

Note that $\mathbb{E}[|\Delta_{k,t-q}|] \leq \mathbb{E}[\tilde{\sigma}_{k,t-q}^2] + \mathbb{E}[\sigma_{k,t-q}^2]$ and this upper bond does not depend on (t, q) by weak stationarity. Thus, letting $q \to \infty$ and under the assumption $a_k + b_k < 1$, we obtain $\mathbb{E}[|\Delta_{k,t}|] = 0$. This implies $\tilde{\sigma}_{k,t} = \sigma_{k,t}$ a.s. for every k. As a consequence, $r_t = \tilde{r}_t$ a.s. because the two solutions share the same hidden state process (s_t), proving the result. The same reasoning can be led for Model (5.6)-(5.7). □

3 Extensions to GARCH(p,q) models

It is natural to extend the latter models in a more general framework. Instead of considering $GARCH(1,1)$ latent processes, we could assume there exist m models $GARCH(p_k, q_k)$, for some positive integers p_k and q_k, $k \in \{1, \ldots, m\}$. With the same m state finite Markov chain $(s_t)_{t\in\mathbb{Z}}$ as above, a general class of switching GARCH models would then be defined by

$$r_t = \mu_{s_t} + \sigma_{s_t,t}\epsilon_{s_t,t}, \text{ where} \tag{5.16}$$

$$\sigma_{k,t}^2 = \omega_k + \sum_{j=1}^{q_k} a_{k,j}\sigma_{k,t-j}^2\epsilon_{k,t-j}^2 + \sum_{j=1}^{p_k} b_k\sigma_{k,t-j}^2, \text{ for any } k \in \{1, \ldots, m\}. \tag{5.17}$$

Still setting $\vec{\epsilon}_t := (\epsilon_{1,t}, \ldots, \epsilon_{m,t})$ in $\mathbb{R}^m$, the process of innovations $(\vec{\epsilon}_t)_{t\in\mathbb{Z}}$ is a sequence of i.i.d. centered random vectors, where $\mathbb{E}[\epsilon_{k,t}^2] = 1$ for any t. Moreover, we still assume that the Markov chain (s_t) and the process of innovations are independent. Note that the components of $\vec{\epsilon}_t$ are dependent in general. They may even all be equal. In the latter case, we recover our model (5.6)-(5.7) above when $p_k = q_k = 1$ for every k. When the components of $\vec{\epsilon}_t$ are mutually independent, Model (5.16)-(5.17) simply becomes Model (5.8)-(5.9) above under MIA, with $p_k = q_k = 1$ for every k.

For the sake of simplicity, we assume all coefficients of our model are nonnegative, insuring that all quantities $\sigma_{k,t}^2$ are nonnegative. This condition is standard and realistic, even if it could be weakened, notably through some $ARCH(\infty)$ representations (see the discussion in Section 2.3.2 in [FZ19], e.g.).

To study the strict stationarity of Model (5.16)-(5.17), let us introduce some notations: for any $k \in \{1, \ldots, m\}$, define the "partially observed" process $(r_{k,t})$ with

$$r_{k,t} := \mu_k + \sigma_{k,t}\epsilon_{k,t}, \ t \in \mathbb{Z}. \tag{5.18}$$

Note that the observable process of interest is $r_t = \sum_{k=1}^{m} 1(s_t = k) r_{k,t}$. Moreover, set the $p_k + q_k$ column vectors

$$\mathbf{b}_{k,t} := \begin{pmatrix} \omega_k \epsilon_{k,t}^2 \\ 0 \\ \vdots \\ 0 \\ \omega_k \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \qquad \mathbf{z}_{k,t} := \begin{pmatrix} (r_{k,t} - \mu_k)^2 \\ \vdots \\ (r_{k,t-q_k+1} - \mu_k)^2 \\ \sigma_{k,t}^2 \\ \vdots \\ \sigma_{k,t-p_k+1}^2 \end{pmatrix}$$

where th component ω_k is located at the $q_k + 1$-th position, and the $(p_k + q_k) \times (p_k + q_k)$ matrix

$$\mathbf{A}_{k,t} := \begin{bmatrix} a_{k,1}\epsilon_{k,t}^2 & & \cdots & & a_{k,q_k}\epsilon_{k,t}^2 & b_{k,1}\epsilon_{k,t}^2 & & \cdots & & b_{k,p_k}\epsilon_{k,t}^2 \\ 1 & 0 & \cdots & & 0 & 0 & & \cdots & & 0 \\ 0 & 1 & \ddots & & 0 & 0 & & \cdots & & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & & \cdots & & \vdots \\ 0 & & \cdots & 1 & 0 & 0 & & \cdots & & 0 \\ a_{k,1} & & \cdots & & a_{k,q_k} & b_{k,1} & & \cdots & & b_{k,p_k} \\ 0 & & \cdots & & 0 & 1 & 0 & \cdots & & 0 \\ 0 & & \cdots & & 0 & 0 & 1 & \ddots & & 0 \\ \vdots & & & & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & \cdots & & 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix}. \tag{5.19}$$

Thus, our model assumptions imply

$$\mathbf{z}_{k,t} = \mathbf{b}_{k,t} + \mathbf{A}_{k,t}\mathbf{z}_{k,t-1},\ t \in \mathbb{Z},\ k \in \{1, \ldots, m\}. \tag{5.20}$$

In other words, the latter relationship is the Markov representation of the k-th underlying $GARCH(p_k, q_k)$ (latent) model. Deduce from (5.20) that

$$\mathbf{z}_{k,t} = \mathbf{b}_{k,t} + \sum_{i=1}^{\infty} \prod_{j=0}^{i-1} A_{k,t-j}\mathbf{b}_{k,t-j},$$

provided that the latter series converges almost surely.

Note that the sequence of random matrices $(\mathbf{A}_{k,t})$ is stationary and ergodic. Thus, we can define its Lyapunov exponent

$$\gamma_k := \lim_{t \to +\infty} \frac{1}{t} \mathbb{E}\big[\ln \|A_{k,t}A_{k,t-1} \ldots A_{k,1}\| \big].$$

Note that γ_k does not depend on the chosen matrix norm because all norms are equivalent here.

The latter Lyapunov exponent satisfies

$$\gamma_k = \lim_{t\to\infty} \frac{1}{t} \ln \|A_{k,t}A_{k,t-1}\dots A_{k,1}\|,$$

almost surely (Theorem 2.3 in [FZ19]).

Remark 4. *In the case of a GARCH(1,1) model, i.e. when $p_k = q_k = 1$ for every k, then the previous Lyapunov exponent γ_k is simply $\mathbb{E}[a_{k,1}\epsilon_{k,1}^2 + b_{k,1}]$, justifying the same notation as in Section 2. Indeed, in this case, note that the 2×2 matrix $\mathbf{A}_{k,t}$ can be rewritten $\mathbf{A}_{k,t} = (\epsilon_{1,t}^2; 1)^\top (a_{k,1}, b_{k,1})$, implying*

$$\mathbf{A}_{k,t}\mathbf{A}_{k,t-1}\dots\mathbf{A}_{k,1} = \prod_{j=1}^{t-1}(a_{k,1}\epsilon_{k,t-j}^2 + b_{k,1})\mathbf{A}_{k,t}.$$

Deduce

$$\frac{1}{t}\mathbb{E}\big[\ln\|\mathbf{A}_{k,t}\mathbf{A}_{k,t-1}\dots\mathbf{A}_{k,1}\|\big] = \frac{1}{t}\mathbb{E}\big[\ln(a_{k,1}\epsilon_{k,t-j}^2 + b_{k,1})\big] + o(1),$$

and the result follows.

Proposition 3.1. *Assume $\gamma_k < 0$ for every $k \in \{1,\dots,m\}$. Then, there exists a strictly stationary solution (r_t) to Model (5.16)-(5.17). When the strictly stationary solution exists, it is unique, non-anticipative and ergodic.*

Proof. Define $\tilde{\mathbf{z}}_{k,t}(N) := \mathbf{b}_{k,t} + \sum_{i=1}^{N}\prod_{j=0}^{i-1} A_{k,t-j}\mathbf{b}_{k,t-j}$. By exactly the same arguments as in the proof of Theorem 2.4 in [FZ19], it can be proved that $\tilde{\mathbf{z}}_{k,t}(N)$ converges a.s. to some random vector $\tilde{\mathbf{z}}_{k,t}$ when $N\to\infty$ that is unique. Denoting $\tilde{\sigma}_{k,t}^2$ the q_k+1-th component of $\tilde{\mathbf{z}}_{k,t}$, the process $(\tilde{r}_{k,t})_{t\in\mathbb{Z}}$ defined by $\tilde{r}_{k,t} := \mu_k + \tilde{\sigma}_{k,t}\epsilon_{k,t}$ is a solution of (5.18). Doing this task for every $k\in\{1,\dots,m\}$ and setting $r_t = \sum_{k=1}^m \mathbf{1}(s_t = k) r_{k,t}$, the process is clearly a solution of (5.16)-(5.17). It is nonanticipative and ergodic, because the m processes $(r_{k,t})_{t\in\mathbb{Z}}$ share this property and they are independent of the hidden states. □

Proposition 3.2. *Assume ω_k and $\mathbb{P}(s_t = k)$ are positive for any $k\in\{1,\dots,m\}$.*

(i) If there exists a second-order stationary and nonanticipative solution of Model (5.16)-(5.17), then

$$\sum_{j=1}^{q_k} a_{k,j} + \sum_{j=1}^{p_k} b_{k,j} < 1,\ k\in\{1,\dots,m\}. \tag{5.21}$$

(ii) Conversely, if (5.21) is satisfied, then the Lyapunov coefficients γ_k are strictly negative and the unique solution of model (5.16)-(5.17) is a weak white noise: for any t, the covariance $\mathbb{E}[r_t r_{t+p}]$ is zero when $p>0$ and

$$\mathbb{E}[r_t^2] = \sum_{k=1}^m \mathbb{P}(s_t = k)\Big\{\frac{\omega_k}{1 - \sum_{j=1}^{q_k} a_{k,j} - \sum_{j=1}^{p_k} b_{k,j}} + \mu_k^2\Big\}. \tag{5.22}$$

Moreover, there exists no other second-order stationary and nonanticipative solution.

Proof. The proof of (i) is similar to the proof of the first point of Proposition 2.3, replacing $a_1 + b_1$ with $\sum_{j=1}^{q_1} a_{1,j} + \sum_{j=1}^{p_1} b_{1,j}$. We omit the details.

Concerning (ii), set $\mathbf{A}_k := \mathbb{E}[\mathbf{A}_{k,t}]$. For any $\lambda \in \mathbb{C}$, it is proved in Lemma 1 in the appendix that

$$\det(\lambda I_{p_k+q_k} - \mathbf{A}_k) = \lambda^{p_k+q_k}\Big(1 - \sum_{j=1}^{q_k} a_{k,j}\lambda^{-j} - \sum_{j=1}^{p_k} b_{k,j}\lambda^{-j}\Big). \tag{5.23}$$

Thus, if $|\lambda| \geq 1$, then

$$\begin{aligned}|\det(\lambda I_{p_k+q_k} - \mathbf{A}_k)| &\geq \Big|1 - \sum_{j=1}^{q_k} a_{k,j}\lambda^{-j} - \sum_{j=1}^{p_k} b_{k,j}\lambda^{-j}\Big| \\ &\geq \quad 1 - \sum_{j=1}^{q_k} a_{k,j}|\lambda^{-j}| - \sum_{j=1}^{p_k} b_{k,j}|\lambda^{-j}| \geq 1 - \sum_{j=1}^{q_k} a_{k,j} - \sum_{j=1}^{p_k} b_{k,j} > 0.\end{aligned}$$

Therefore, the modulus of any eigenvalue of $\mathbf{A}_k$ is strictly smaller than one. In other words, denoting $\rho(\mathbf{A}_k)$ the spectral radius of $\mathbf{A}_k$, this means $\rho(\mathbf{A}_k) < 1$. Invoking Equation (1.4) in [KS84], this yields

$$\gamma_k \leq \ln\left(\rho(\mathbf{A}_k)\right) < 0, \ k \in \{1, \ldots, m\}.$$

Due to Proposition 3.1, there exists a unique strictly (and then weak) stationary solution of Model (5.16)-(5.17). Its mean is zero because the solution is nonanticipatory. Moreover, taking the expectation of (5.17), we get

$$\mathbb{E}[\sigma_{k,t}^2] = \omega_k/(1 - \sum_{j=1}^{q_k} a_{k,j} - \sum_{j=1}^{p_k} b_{k,j}),$$

for any $k \in \{1, \ldots, m\}$. Moreover, we have

$$\begin{aligned}\mathbb{E}[r_t^2] &= \sum_{k=1}^{m} p_k \mathbb{E}[r_t^2 | s_t = k] = \sum_{k=1}^{m} p_k \mathbb{E}[(\mu_k + \sigma_{k,t}\epsilon_t)^2] \\ &= \sum_{k=1}^{m} p_k (\mathbb{E}[\sigma_{k,t}^2]\mathbb{E}[\epsilon_t^2] + \mu_k^2 + 2\mu_k \mathbb{E}[\sigma_{k,t}]\mathbb{E}[\epsilon_t]) \\ &= \sum_{k=1}^{m} p_k (\mathbb{E}[\sigma_{k,t}^2] + \mu_k^2),\end{aligned}$$

yielding (5.22).

The uniqueness of the solution can be proved as in Proposition 2.3, but requires more care. First, for every k, denote $r_k = \max(p_k, q_k)$. Set $a_k = 0$ (resp. $b_k = 0$) when $k > q_k$ (resp. $k > p_k$), and define the maps $f_{k,j}(\epsilon) := a_{k,j}\epsilon^2 + b_{k,j}$, $\epsilon \in \mathbb{R}$, and $j \in \{1, \ldots, r_k\}$. Moreover, denote the r_k-dimensional vectors $\mathbf{v}_{k,t} := [\sigma_{k,t}^2, \ldots, \sigma_{k,t-r_k+1}^2]$ and $\vec{\omega}_k := [\omega_k, 0, \ldots, 0]$. Then,

$\mathbf{v}_{k,t} = \vec{\omega}_k + M_{k,t}\mathbf{v}_{t-1}$ for every t by (5.17), introducing the random matrices

$$M_{k,t} := \begin{bmatrix} f_{k,1}(\epsilon_{k,t-1}) & f_{k,2}(\epsilon_{k,t-2}) & \cdots & \cdots & f_{k,r_k}(\epsilon_{k,t-r_k}) \\ 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix}.$$

Assume there exists another second-order stationary and nonanticipative solution $(\tilde{r}_t)_{t\in\mathbb{Z}}$ of model (5.16)-(5.17). Denote $\tilde{\sigma}_{k,t}$ its corresponding instantaneous volatilities and $\tilde{\mathbf{v}}_{k,t} := [\tilde{\sigma}^2_{k,t}, \ldots, \tilde{\sigma}^2_{k,t-r_k+1}]$. As for the processes $(\mathbf{v}_{k,t})$, they satisfy $\tilde{\mathbf{v}}_{k,t} = \vec{\omega}_k + M_{k,t}\tilde{\mathbf{v}}_{k,t-1}$. Setting $\Delta_{k,t} := \tilde{\mathbf{v}}_{k,t} - \mathbf{v}_{k,t}$, we deduce

$$\Delta_{k,t} = M_{k,t}\Delta_{t-1} = \ldots = \prod_{j=1}^{q} M_{k,t}\Delta_{k,t-q},\ q > 1.$$

Since the matrices $M_{k,t}$ are nonnegative, we obtain the componentwise inequality

$$|\Delta_{k,t}| \leq \prod_{j=1}^{q} M_{k,t}|\Delta_{k,t-q}|,\ q > 1.$$

Taking the expectation of the latter vector componentwise and recalling that the solution is nonanticipative, we get

$$\mathbb{E}[|\Delta_{k,t}|] \leq \mathbb{E}[\prod_{j=1}^{q} M_{k,t-j}]\mathbb{E}[|\Delta_{k,t-q}|],$$

still componentwise. It can be proved (see below) that

$$\mathbb{E}[\prod_{j=1}^{q} M_{k,t-j}] = \prod_{j=1}^{q} \mathbb{E}[M_{k,t}], \tag{5.24}$$

even if $(M_{k,t})_{t\in\mathbb{Z}}$ is not a sequence of independent random matrices. Since $\mathbb{E}[M_{k,t-j}]$ does not depend on t, it is denoted M_k and we get $\mathbb{E}[\prod_{j=1}^{q} M_{k,t-j}] = M_k^q$.

The characteristic polynomial of M_k is

$$\det(\lambda I_{r_k} - M_k) = \lambda^{r_k} - \sum_{j=1}^{r_k} \lambda^{r_k-j}(a_{k,j} + b_{k,j}) = \lambda^{r_k}\{1 - \sum_{j=1}^{r_k} \lambda^{-j}(a_{k,j} + b_{k,j})\}.$$

Thus, an eigenvalue λ_0 of M_k is zero or satisfies $1 = \sum_{j=1}^{r_k} \lambda_0^{-j}(a_{k,j} + b_{k,j})$. If $|\lambda_0| \geq 1$, then

$$1 = |\sum_{j=1}^{r_k} \lambda_0^{-j}(a_{k,j} + b_{k,j})| \leq \sum_{j=1}^{r_k} |\lambda_0^{-j}|(a_{k,j} + b_{k,j}) \leq \sum_{j=1}^{r_k} (a_{k,j} + b_{k,j}),$$

which contradicts (5.21). Thus, $\rho(M_k) < 1$. For any deterministic matrix M and and any matrix norm $\|\cdot\|$, it is known that $\rho(M)$, the spectral radius of M, satisfies $\rho(M) = \lim_{t\to+\infty} \|M^t\|^{1/t}$ (Gelfand formula, Corollary 5.6.14 in [HJ12]). This implies $M_k^q \to 0$ when $q \to \infty$ ([Lüt97], Sec-

tion 5.4). Note that every component of $\mathbb{E}[|\Delta_{k,t-q}|]$ is not larger than $\max_{j=0,\dots,r_k-1}\{\mathbb{E}[\tilde{\sigma}^2_{k,t-q-j}]+\mathbb{E}[\sigma^2_{k,t-q-j}]\}$. The latter upper bond is finite and does not depend on (t,q) by the second-order stationarity of the two solutions. Thus, letting $q\to\infty$, we obtain $\mathbb{E}[|\Delta_{k,t}|]=0$. This implies $\tilde{\sigma}_{k,t}=\sigma_{k,t}$ a.s. for every k. As a consequence, $r_t=\tilde{r}_t$ a.s. because the two solutions share the same hidden state proces (s_t), proving the uniqueness of second-order stationary solutions.

It remains to prove (5.24). For any $j\in\mathbb{N}$, let $\mathcal{F}_{t:t-j}$ be the set of random variables that can be written as sums of random elements

$$f_1(\epsilon^2_{k,t-i_1})f_2(\epsilon^2_{k,t-i_2})\cdots f_N(\epsilon^2_{k,t-i_N}),\ N\in\mathbb{N}^*,$$

for some distinct indices $i_1,\dots,i_N$ in $\{0,1,\dots,j\}$, and some affine univariat maps f_i, $i\geq 1$: for any i, there exists some nonnegative constants α_i and β_i s.t. $f_i(t)=\alpha_i+\beta_i t$, $t\in\mathbb{R}$. We will prove by induction on q the following property:

"$\mathcal{H}_q$: the components of the ℓ-th column vector of $M_{k,t}M_{k,t-1}\cdots M_{k,t-q}$ belong to $\mathcal{F}_{t:t-q-\ell+1}$, for any $\ell\in\{1,\dots,r_k\}$."

When $q=0$, $\mathcal{H}_0$ is clearly satisfied by inspecting the components of $M_{k,t}$. Now, assume $\mathcal{H}_q$ is true for some $q\in\mathbb{N}$ and let us check $\mathcal{H}_{q+1}$. Let us inspect $M_{k,t}M_{k,t-1}\cdots M_{k,t-q-1}$. In its first column, the elements are sums of products of some elements in $\mathcal{F}_{t:t-q}$ and $a_{k,1}\epsilon^2_{k,t-q-1}+b_{k,1}$ or one. Then, the result belongs to $\mathcal{F}_{t:t-q-1}$. Similarly, the elements in its second column are given by the product of some elements in $\mathcal{F}_{t:t-q-1}$ and $a_{k,2}\epsilon^2_{k,t-q-2}+b_{k,2}$ (or one), and their sum belongs to $\mathcal{F}_{t:t-q-2}$, etc., by reasoning column per column. This finally yields $\mathcal{H}_{q+1}$, and then the property $\mathcal{H}_q$ is true for any q.

Now, the expectation of $M_{k,t}M_{k,t-1}\cdots M_{k,t-q}$ can be directly evaluated componentwise. Indeed, with our previous notations, the expectation of some elements of the latter matrix is a sum of terms as

$$\mathbb{E}[f_1(\epsilon^2_{k,t-i_1})f_2(\epsilon^2_{k,t-i_2})\cdots f_N(k,\epsilon^2_{t-i_N})]=f_1(1)f_2(1)\cdots f_N(1),$$

by independence between our innovations. Thus, the expectation of $M_{k,t}M_{k,t-1}\cdots M_{k,t-q}$ is obtained by the product of the latter matrices as if all innovations were constant and are equal to their mean 1. This is equivalent to calculating $\mathbb{E}[M_{k,t}]\mathbb{E}[M_{k,t-1}]\cdots\mathbb{E}[M_{k,t-q}]$, proving (5.24). □

4 Estimation by Nonparametric Simulated Maximum Likelihood

We observe a sample path $(r_1,r_2,\dots,r_T)$ from our new model (5.8)-(5.9), i.e.

$$r_t=\mu_{s_t}+\sigma_{s_t,t}\epsilon_{s_t,t},\ \text{ and} \tag{5.25}$$

$$\sigma^2_{k,t}=\omega_k+a_k\sigma^2_{k,t-1}\epsilon^2_{k,t-1}+b_k\sigma^2_{k,t-1},\ \text{for any } k\in\{1,\dots,m\}. \tag{5.26}$$

We obviously impose $\mathbb{E}[\epsilon_{k,t}] = 0$ and $\mathbb{E}[\epsilon_{k,t}^2] = 1$ for any k and any t. Our second model (5.6)-(5.7) will correspond to the particular case when $\epsilon_{k,t} = \epsilon_{1,t}$, for every $k \in \{2, \ldots, m\}$ and every t. It has to be observed that the calculation of the likelihood is untractable due to path dependencies, contrary to [HMP04]. This situation cannot be solved by Hamilton filter and the methodology detailed in Appendix 7.1 because the $\sigma_{k,t}$ cannot be analytically calculated (they potentially depend on all latent noises $\epsilon_{k',t'}$, $k' \neq k$, $t' \leq t$). Thus, the inference task seems to be hard.

We propose to solve the problem by Nonparametric Simulated Maximum Likelihood. This is an omnibus simulation-based inference procedure initially proposed in [FS04] to estimate static models for which likelihoods cannot be analytically evaluated. [KS12] generalized the latter methodology to dynamic models, our situation in this paper. See similar ideas in [AM09], [CK12], among others.

To be specific, assume that the Data Generating Process of the vector of innovations $\vec{\epsilon}_t := (\epsilon_{1,t}, \ldots, \epsilon_{m,t})$ is known. Note that we do not impose independence between the components of the random vector of innovations. Typically, the law of $\vec{\epsilon}_t$ could be standard Gaussian $\mathcal{N}(0_m, Id_m)$. Nonetheless, to be in line with [KS12], we assume the vectors of innovations are i.i.d. and drawn from a parametric law $F_\epsilon(\zeta)$, that potentially depends on an unknown vector of parameters ζ. Our global vector of parameters is then

$$\theta = (\zeta; \mu_k, \omega_k, a_k, b_k, k \in \{1, \ldots, m\}; p_{i,j}, i < j),$$

and the true parameter is denoted θ_0. We assume the model is identifiable. This property has to be checked for every particular choice of innovation laws. To avoid non-identifiability due to a re-ordering of our state indices, it is necessary to impose additional constraints. For instance, assuming $\mu_1 < \mu_2 < \ldots < \mu_m$ would be a logical and intuitive choice in terms of interpretation. Alternatively, an order of unconditional variance could be imposed, i.e. $\omega_1/(1 - a_1 - b_1) < \omega_2/(1 - a_2 - b_2) < \ldots < \omega_m/(1 - a_m - b_m)$. Moreover, as for usual GARCH(1,1) processes, it is necessary to dismiss degenerate innovation processes by assuming that the laws of the $\epsilon_{k,t}$ are never Dirac.

Instead of trying to analytically evaluate the log-likelihood of the observations $\mathcal{L}^*(\theta) := \sum_{t=1}^T f(r_t | r_{t-1}, \ldots, r_1)$, we can focus on some approximations of the latter map by considering only a finite number of lagged values, as already proposed in [FS04] or [KS12]: for some integer q, set the likelihood-type criterion

$$\mathcal{L}_q^*(\theta) := \sum_{t=q+1}^{T} f(r_t | r_{t-1}, \ldots, r_{t-q}). \tag{5.27}$$

When q is large, optimizing $\mathcal{L}_q^*(\theta)$ is close to optimizing $\mathcal{L}^*(\theta)$, hopefully. But even if q is small, optimizers of $\mathcal{L}_q^*(\theta)$ converge to θ_0 in probability under some identification and regularity conditions. This is the same situation as for composite likelihood techniques, and we will promote $q \leq 3$ typically. The latter choice is most often sufficient to ensure identifiability of the model by the knowledge of conditional univariate laws given q lagged values.

In our situation $f(r_t|r_{t-1},\dots,r_{t-q})$ is untractable and will be estimated by simulation: for a given parameter value θ, simulate a large number M of trajectories $\big(\vec{\epsilon}_{t',i}(\theta)\big)_{t'=1,\dots,T}$, $i \in \{1,\dots,M\}$, given our model equations (5.25)-(5.26) and knowing $F_\epsilon(\gamma)$. We obtain a sample of M simulated vectors $\big(r_{t,i}(\theta), r_{t-1,i}(\theta),\dots,r_{t-q,i}(\theta)\big)$, $i \in \{1,\dots,M\}$ at every date t. These vectors allow us to nonparametrically estimate $f(r_t|r_{t-1},\dots,r_{t-q})$ by

$$\hat{f}(r_t|r_{t-1},\dots,r_{t-q};\theta) := \frac{\sum_{i=1}^M K_{h,q+1}\big(r_t - r_{t,i}(\theta), r_{t-1} - r_{t-1,i}(\theta),\dots,r_{t-q} - r_{t-q,i}(\theta)\big)}{\sum_{i=1}^M K_{h,q}\big(r_{t-1} - r_{t-1,i}(\theta),\dots,r_{t-q} - r_{t-q,i}(\theta)\big)}, \tag{5.28}$$

where, for any p-dimensional kernel $K_p : \mathbb{R}^p \to \mathbb{R}$, $\int K_p = 1$, $p \geq 1$, and a p-dimensional vector $h = (h_1,\dots,h_p)$ of positive numbers, we denote

$$K_{p,h}(x_1,\dots,x_p) := K(x_1/h_1,\dots,x_p/h_p)/(h_1\cdots h_p),\ x_j \in \mathbb{R}, j \in \{1,\dots,p\}.$$

Obviously, (5.28) is the usual Nadaraya-Watson kernel estimate of a conditional density. Other nonparametric regressors could surely be invoked, even if the associated theoretical guarantees have to be stated in the literature, to the best of our knowledge.

Therefore, our NPSML estimator of θ_0 is defined as

$$\hat{\theta}_q := \arg\max_\theta \sum_{t=q+1}^T \tau_a\big(\hat{f}(r_t|r_{t-1},\dots,r_{t-q};\theta)\big) \ln\big(\hat{f}(r_t|r_{t-1},\dots,r_{t-q};\theta)\big), \tag{5.29}$$

where $\tau_a(\cdot)$ is a trimming map that avoids numerical issues when the estimated conditional density is close to zero. Typically, τ_a is continuously differentiable on $\mathbb{R}$ s.t. $\tau_a(t) = 1$ when $|t| > a$ and $\tau_a(t) = 0$ when $|t| < a/2$, with a tuning parameter a that tends to zero with T [1]. See [KS12] for theoretical results related to the consistency and the asymptotic normality of such estimators $\hat{\theta}_q$.

Note that other criteria could be proposed. For instance, [FS04] proposed to refer to the joint laws of (r_t, r_{t-k}), for some $k \in \{1,\dots,q\}$, instead of focusing on the conditional law of r_t given $(r_{t-1},\dots,r_{t-q})$. With the same notations as above, an estimator of $f(r_t, r_{t-k})$ could be

$$\hat{f}(r_t, r_{t-k};\theta) := \frac{1}{M}\sum_{i=1}^M K_{h,2}\big(r_t - r_{t,i}(\theta), r_{t-k} - r_{t-k,i}(\theta)\big),\ k \geq 1.$$

The associated estimator would be

$$\hat{\theta}_{b,k} := \arg\max_\theta \sum_{t=k+1}^T \tau_a\big(\hat{f}(r_t, r_{t-k};\theta)\big) \ln\big(\hat{f}(r_t, r_{t-k};\theta)\big). \tag{5.30}$$

More generally, taking more bivariate distributions into account would increase the efficiency of

1. For instance, set $\tau_a(t) = \mathbf{1}(t \in [a/2, a])(t - a/2)^2(20 - 16t/a)/a^2 + \mathbf{1}(t > a)$.

the procedure. This prompts us to propose

$$\hat{\theta}_{b,1:p} := \arg\max_{\theta} \sum_{k=1}^{p} \sum_{t=k+1}^{T} \tau_a(\hat{f}(r_t, r_{t-k}; \theta)) \ln\left(\hat{f}(r_t, r_{t-k}; \theta)\right). \tag{5.31}$$

Straightforward variations of the latter ideas with trivariate, or even higher dimensional, distributions are possible. Nonetheless, nonparametrically evaluating s-dimensional multivariate densities becomes numerically challenging when s is larger than three, typically, due to the so-called "curse of dimension". Thus, in our opinion, working with $\hat{\theta}_q$ when $q \leq 2$ or with $\hat{\theta}_{b,1:p}$ for any p is reasonable.

Note that the inference strategy detailed in this section applies for any joint law of the innovations $F_\epsilon(\gamma)$, in particular when it is degenerate i.e. $\epsilon_{1,t} = \epsilon_{2,t} = \ldots = \epsilon_{m,t}$ almost surely. In such a case, the underlying model is given by (5.6) and (5.7).

5 Empirics

Let us study by simulation the empirical performances of the inference procedures proposed in Section 4, in the bivariate case ($m = 2$ with our notations). To fix the ideas, let us choose standard Gaussian innovations (case A) or perfectly positively dependent standard Gaussian innovations (case B). In the former case, the vectors of innovations $(\epsilon_{1,t}, \epsilon_{2,t})_{t\geq 0}$ are independent, and $\mathbb{E}[\epsilon_{1t}\epsilon_{2t}] = \rho$, for some $\rho \in (-1, 1)$. In the latter case, $\epsilon_{1t} = \epsilon_{2t} =: \epsilon_t$ for every t. Set $\mu_1 = 1$ and $\mu_2 = (-1)$, $\omega_1 = 1$ and $\omega_2 = 2$. For identification purpose, we will impose $\omega_1 < \omega_2$, a constraint that will be put in the optimizer. We impose two latent stationary GARCH dynamics by choosing (a_k, b_k) such that $0 < a_k + b_k < 1$, $k \in \{1, 2\}$. The transition matrix will be defined by $p_{1,2}$ and $p_{2,1}$. For instance, set $p_{1,2} = 0.05$ and $p_{2,1} = 0.2$. This means that state 1 is more persistent than state 2. It will be associated to smaller average variances when $\omega_1/(1 - a_1 - b_1) < \omega_2/(1 - a_2 - b_2)$, a condition that will be imposed when (a_1, a_2, b_1, b_2) will be drawn.

The initial values of the simulated processes are given by $\epsilon_{k,0} = 0$ and $\sigma_{k,0} = 1$, $k \in \{1, 2\}$, in the case of the "observed" $(r_t)_{t=1,\ldots,T}$ as for the simulated trajectories $\left(r_{t,i}(\theta)\right)_{t=1,\ldots,T}$. To weaken the influence of the first values in such samples, the first 100 are "burned" (i.e. forgotten) in the calculations of our criteria.

We will use usual multivariate Gaussian kernels

$$K_p(x_1, \ldots, x_p) = (2\pi)^{-p/2} \exp\Big(-\sum_{j=1}^{p} x_j^2/2\Big).$$

For a reasonable value for M ($M = 200$ here), we calculated the estimated vector of parameters

$$\theta = (\mu_0, \mu_1, \omega_0, \omega_1, \alpha_0, \alpha_1, \beta_0, \beta_1, p_{00}, p_{11}).$$

As a measure of performance, we select the L^1-type relative value

$$RV_1 := \frac{1}{|\theta|_0} \sum_{j=1}^{|\theta|_0} |\hat{\theta}_j - \theta_{0,j}|/|\theta_{0,j}|.$$

We repeated the same procedure N times. We calculate the average RV_1 over all the $N = 200$ experiments, i.e.

$$MRV_1 := \mathbb{E}[\|\hat{\theta} - \theta_0\|_1/\|\theta_0\|_1] \simeq \frac{1}{N} \sum_{k=1}^{N} RV_{1,k},$$

and similarly for medians, quantiles, etc.

	μ_0	μ_1	ω_0	ω_1	α_0	α_1	β_0	β_1	p_{00}	p_{11}
parameters	-1	1	1	2	0.2	0.1	0.7	0.5	0.95	0.8
count	200	200	200	200	200	200	200	200	200	200
mean	0.1956	0.2925	0.1599	0.0917	0.4130	0.6292	0.1894	0.2135	0.0847	0.1283
std	0.1882	0.3836	0.1578	0.0931	0.3317	0.8146	0.1695	0.1637	0.0804	0.0988
min	0.0026	0.0022	0.0018	0.0002	0.0011	0.0000	0.0000	0.0004	0.0006	0.0012
5%	0.0164	0.0186	0.0097	0.0080	0.0258	0.0000	0.0000	0.0174	0.0077	0.0084
25%	0.0721	0.0597	0.0480	0.0327	0.1527	0.0073	0.0683	0.0859	0.0288	0.0503
50%	0.1529	0.1590	0.1099	0.0699	0.3566	0.1396	0.1333	0.1711	0.0539	0.0991
75%	0.2537	0.2959	0.2172	0.1191	0.4997	1.0993	0.2718	0.3342	0.1028	0.2060
95%	0.5072	1.3173	0.4988	0.2553	1.1585	2.3635	0.5001	0.5169	0.2590	0.3230
max	1.6173	1.8261	0.7900	0.7500	1.4674	2.8716	0.8571	0.8000	0.4737	0.3750

Table 5.1 – Summary statistics for parameter estimation for 200 simulations

Table 5.1 shows some statistics related to the distribution of the L^1-type relative value errors for the set of parameters, obtained through 200 simulations. Focusing on median, it provides us an insight about the central tendency of estimation performance. Looking at the parameters μ_0 and μ_1, our results exhibit similar median errors (0.1529 and 0.1590, respectively), indicating a comparable estimation accuracy. Concerning ω_0 and ω_1, we got median relative errors equal to 0.0990 and 0.0699, respectively, which suggests a higher estimation accuracy. In contrast, the parameters α_0 and α_1 have higher median errors, 0.3566 and 0.1396. It indicates an higher difficulty to perform an accurate estimation. The parameters β_0 and β_1 show moderate median errors, 0.1333 and 0.1711, respectively. This reflects moderate estimation accuracy. Alternatively, the probability p_{00} exhibits the lowest median error of 0.0539 and then the highest estimation accuracy among all parameters. Overall, the estimation technique seems to perform rather well.

6 Conclusion

To conclude, the proposed new MS-GARCH model is meaningful and the performances of our estimation strategy is promising. Its ability to handle parameters related to regime-switching probabilities and basic volatility terms is good overall. However, relatively high uncertainties for some parameters indicate areas where the model could be refined. The accuracy of our procedure could possibly be improved with additional constraints. This analysis highlights the importance of continuous evaluation and adaptation of MS-GARCH models to ensure robust and reliable parameter estimation in financial modelling.

7 Appendix

7.1 Calculation of a likelihood function for the Haas et al. (2004) model with non zero conditional means

We observe a sequence of asset returns $r_1, \dots, r_T$. Denote by $\mathcal{F}_t$ the observations until time t, i.e. $r_1, \dots, r_t$. Assume the underlying MS-GARCH model is given by

$$r_t = \mu_{s_t} + \sigma_{s_t,t}\epsilon_t,\ \mathbb{E}[\epsilon_t|\mathcal{F}_{t-1}] = 0,\ \mathbb{E}[\epsilon_t^2|\mathcal{F}_{t-1}] = 1, \tag{5.32}$$

where

$$\sigma_{k,t}^2 = \omega_k + a_k(r_{t-1} - \mu_k)^2 + b_k\sigma_{k,t-1}^2, \tag{5.33}$$

for any $k \in \{1, \dots, m\}$. The innovations $\epsilon_1, \dots, \epsilon_T$ are i.i.d. The density of ϵ_t is denoted f_ϵ.

Denote $p_{i,j} := \mathbb{P}(s_t = j|s_{t-1} = i)$, when $i, j \in \{1, \dots, m\}$ the unknown transition probabilities. The vector of parameters is denoted as θ. It stacks all subvectors (ω_k, a_k, b_k), $k \in \{1, \dots, m\}$, the vector of conditional means $(\mu_1, \dots, \mu_m)$, the unknown probabilities $p_{i,j}$ when $i < j$ and, possibly, some additional parameters that define the law of ϵ_t. Thus, given the past until time $t-1$, i.e. given $\mathcal{F}_{t-1}$, the law of the asset return at time t depends on the state s_t. The latter state is unobservable and determines the conditional mean μ_{s_t} of r_t and its conditional variance σ_t (Equation (5.32)).

The conditional density of r_t given its past values can be written

$$\begin{aligned} f(r_t|\mathcal{F}_{t-1}) &= \sum_{i,j=1}^{m} f(r_t, s_t = j, s_{t-1} = i|\mathcal{F}_{t-1}) \\ &= \sum_{i,j=1}^{m} f(r_t|s_t = j, s_{t-1} = i, \mathcal{F}_{t-1})\mathbb{P}(s_t = j, s_{t-1} = i|\mathcal{F}_{t-1}) \\ &= \sum_{i,j=1}^{m} p_{i,j} f(r_t|s_t = j, \mathcal{F}_{t-1})\mathbb{P}(s_{t-1} = i|\mathcal{F}_{t-1}), \end{aligned} \tag{5.34}$$

when $t \in \{1, \dots, T\}$. We will set $\mathcal{F}_0 = \emptyset$ and $f(r_1|\mathcal{F}_0) = f(r_1)$, where f denotes the unconditional distribution of r_1. In other words,

$$f(r_1|\mathcal{F}_0) = f_\epsilon((r_1 - \mathbb{E}[\mu_{s_t}])/\mathbb{E}[\sigma_{s_t,0}])/\mathbb{E}[\sigma_{s_t,0}].$$

In practice, we will simply replace $\mathbb{E}[\mu_{s_t}]$ (respectively $\mathbb{E}[\sigma_{s_t,0}]$) by the empirical mean (resp. standard deviation) of (r_t).

The goal is to calculate (5.34) for every $t \in \{1, \dots, T\}$, and then the log-likelihood. Indeed, the log-likelihood of the sample will be

$$\mathcal{L}(\theta) := \sum_{t=1}^{T} \ln f(r_t|\mathcal{F}_{t-1}),$$

and the latter function has to be optimized with respect to its parameter θ.

First, calculate $f(r_t|s_t = j, \mathcal{F}_{t-1})$ through the law of ϵ_t, once $\sigma_{j,t}$ is evaluated. This latter point is done by recursively using (5.33) backwards, from t to the initial date. To be more specific, denote by f_ϵ the density of ϵ_t. Thus, we have

$$f(r_t|s_t = j, \mathcal{F}_{t-1}) = f_\epsilon\big((r_t - \mu_j)/\sigma_{j,t}\big)/\sigma_{j,t}, \tag{5.35}$$

and the calculation of $\sigma_{j,t}$ can be made iteratively:

$$\begin{aligned}
\sigma^2_{j,t} &= \omega_j + a_j(r_{t-1} - \mu_j)^2 + b_j\sigma^2_{j,t-1} \\
&= \omega_j(1 + b_j) + a_j(r_{t-1} - \mu_j)^2 + b_j a_j(r_{t-2} - \mu_j)^2 + b_j^2\sigma^2_{j,t-2} \\
&= \frac{\omega_j(1 - b_j^t)}{1 - b_j} + \sum_{k=1}^{t} a_j b_j^{k-1}(r_{t-k} - \mu_j)^2 + b_j^t\sigma^2_{j,0}.
\end{aligned}$$

In (5.34), the quantity $\mathbb{P}(s_{t-1} = i|\mathcal{F}_{t-1})$ can be calculated by the Hamilton filter (see [Ham20], Chapter 22), that we recall now: introduce the notations $\zeta_{i,t|t} := \mathbb{P}(s_t = i|\mathcal{F}_t)$ and $\zeta_{i,t|t-1} := \mathbb{P}(s_t = i|\mathcal{F}_{t-1})$. Thus, we have

$$\begin{aligned}
\zeta_{i,t|t} = \mathbb{P}(s_t = i|\mathcal{F}_t) &= \frac{\mathbb{P}(s_t = i, r_t|\mathcal{F}_{t-1})}{f(r_t|\mathcal{F}_{t-1})} \\
&= \frac{f(r_t|s_t = i, \mathcal{F}_{t-1})\mathbb{P}(s_t = i|\mathcal{F}_{t-1})}{\sum_{k=1}^m f(r_t|s_t = k, \mathcal{F}_{t-1})\mathbb{P}(s_t = k|\mathcal{F}_{t-1})} \\
&= \frac{f(r_t|s_t = i, \mathcal{F}_{t-1})\zeta_{i,t|t-1}}{\sum_{k=1}^m f(r_t|s_t = k, \mathcal{F}_{t-1})\zeta_{k,t|t-1}}.
\end{aligned} \tag{5.36}$$

But, note that

$$\zeta_{k,t|t-1} = \sum_{l=1}^{m} p_{kl}\zeta_{l,t-1|t-1},$$

and $f(r_t|s_t = i, \mathcal{F}_{t-1})$ can be calculated as in (5.35). This yields

$$\zeta_{i,t|t} = \mathbb{P}(s_t = i|\mathcal{F}_t) = \frac{f(r_t|s_t = i, \mathcal{F}_{t-1})\sum_{l=1}^m p_{kl}\zeta_{l,t-1|t-1}}{\sum_{k,l=1}^m f(r_t|s_t = k, \mathcal{F}_{t-1})p_{kl}\zeta_{l,t-1|t-1}}. \tag{5.37}$$

Finally, (5.36) allows to recursively calculate the quantities $\zeta_{i,t|t}$, and then the conditional density of r_t given its the past values.

The estimator of θ is then

$$\hat{\theta} := \arg\max_{\theta\in\Theta} \mathcal{L}(\theta) = \arg\max_{\theta\in\Theta} \sum_{t=1}^{T} \ln f(r_t|\mathcal{F}_{t-1}), \tag{5.38}$$

for some subset closed and bounded Θ.

7.2 Path dependencies in MS-GARCH models

In this section, we explain the difficulty of estimating a Markov-Switching GARCH model by maximum likelihood (the main universal estimation procedure), starting from the realization of a time series $(z_t)_{t=1,\dots,T}$.

To fix the ideas, first consider the simplest Markov-Switching model as possible: there exists an underlying homogenous Markov chain (s_t), $s_t \in \{1,2\}$ and $z_t \sim \mathcal{N}(\mu_{s_t}, 1)$, for every $t \in \{1,\dots,T\}$. The parameter θ to be estimated is the concatenation of μ_1, μ_2 and two transition probabilities $p_{1,2}$ and $p_{2,1}$. The log-likelihood associated to the observed path and the latter model is $\mathcal{L}(\theta) = \sum_{t=1}^{T} \ln f(z_t|z_{t-1},\dots,z_1)$ where $f(z_t|z_{t-1},\dots,z_1)$ is the conditional density of z_t given its past values. Since s_t is unknown, we have

$$\begin{aligned} f(z_t|z_{t-1},\dots,z_1) &= \sum_{j=1}^{2} \mathbb{P}(s_t = j|z_{t-1},\dots,z_1) f(z_t|s_t = j, z_{t-1},\dots,z_1) \\ &= \sum_{j=1}^{2} \mathbb{P}(s_t = j|z_{t-1},\dots,z_1)\phi(z_t - \mu_j), \end{aligned}$$

denoting by ϕ the density of a standard Gaussian random variable.

By the so-called Hamilton filter, it is possible to explicitly and iteratively calculate the quantities $\mathbb{P}(s_t = j|z_{t-1},\dots,z_1)$. Thus, the calculation of $\mathcal{L}(\theta)$ can effectively be made; in particular, the latter map can easily be optimized with respect to θ, to obtain the maximum likelihood estimator of the model parameter.

In the case of MS-GARCH models, for which the current volatility at time t depends on the states s_t and on the whole path $(z_j)_{j\le t}$, things become more complex. Now, assume a very simple MS-GARCH model: for every t, $z_t \sim \mathcal{N}(0, \sigma^2_{t,s_t})$ with the variance dynamics

$$\sigma^2_{t,j} = \omega_j + a_j r^2_{t-1} + b_j \sigma^2_{t-1,s_{t-1}},\ j \in \{1,2\}. \tag{5.39}$$

The new vector of parameters becomes $\theta := (\omega_1, \omega_2, a_1, a_2, b_1, b_2, p_{1,2}, p_{2,1})$. The calculation of the associated log-likelihood, or, equivalently, of the conditional densities $f(z_t|z_{t-1},\dots,z_1)$ will induce the so-called "path dependency problem". Indeed, $f(z_t|z_{t-1},\dots,z_1)$ depends on σ_{t,s_t}, that is not observable and depends itself on all the past states (not only s_t). As a consequence, we have

$$\begin{aligned} f(z_t|z_{t-1},\dots,z_1) = \sum_{s_1=1}^{2} \cdots \sum_{j_t=1}^{2} &\mathbb{P}(s_1 = j_1,\dots,s_t = j_t|z_{t-1},\dots,z_1) \\ &f(z_t|s_1 = j_1,\dots,s_t = j_t, z_{t-1},\dots,z_1). \end{aligned}$$

The latter formula cannot be calculated analytically in practice because it involves 2^T terms, i.e. an exploding number of terms. Therefore, we are facing a problem of computational statistics [2].

2. This is explained slightly differently in [Gra96]: *"The conditional variance at time t depends upon the conditional variance at time $t-1$, which depends upon the regime at time $t-1$ and on the conditional variance at*

The first attempts to solve such "path dependencies" were due to [Cai94] and [HS94]. The latter authors introduced some regime-switching models in which the conditional variances in each regime depend on a finite and reduced number of past variances and squared returns. This means considering ARCH processes instead of GARCH processes, and restricting "memory" to a finite number of past periods. This is clearly restrictive for financial returns. Indeed, it is widely recognized that they exhibit long-memory patterns.

Another proposition was due to [Gra96] that apparently slightly modified the MS-GARCH dynamics to circumvent path dependencies. In his paper, he replaced (5.39) by

$$\sigma_{t,j}^2 = \omega_j + a_j r_{t-1}^2 + b_j\{p_{1,t-1}\sigma_{t-1,1}^2 + p_{2,t-1}\sigma_{t-1,2}^2\}, \text{ where} \tag{5.40}$$

$$p_{j,t-1} = \mathbb{P}(s_{t-1} = j | z_{t-2}, \ldots, z_1),\ j \in \{1, 2\}.$$

The quantities $p_{j,t-1}$ can be iteratively calculated by Hamilton filter. Under the new dynamics (5.40), each conditional variance depends only on the current regime, not on the entire past history of the process, breaking the path dependency curse. The associated tree is said to be "recombining". Note that the term $p_{1,t-1}\sigma_{t-1,1}^2 + p_{2,t-1}\sigma_{t-1,2}^2$ is the conditional expectation of $\sigma_{t-1,s_{t-1}}^2$ given the information until $t-2$ in terms of returns and states (or, equivalently given r_{t-2} because (s_t) is a homogenous Markov chain).

Alternatively, [Kla02] promoted the same type of model as [Gra96], but by modifying the way of calculating the average variances in the updating equations:

$$\sigma_{t,j}^2 = \omega_j + a_j r_{t-1}^2 + b_j\{q_{1,t-1}\sigma_{t-1,1}^2 + q_{2,t-1}\sigma_{t-1,2}^2\}, \text{ where} \tag{5.41}$$

$$q_{j,t-1} = \mathbb{P}(s_{t-1} = j | s_t = j, z_{t-1}, \ldots, z_1),\ j \in \{1, 2\}.$$

Note the difference between $q_{j,t-1}$ and the previous quantity $p_{j,t-1}$. Thus, [Kla02] uses more information for the purpose of average variance calculations, say the knowledge of the current state s_t and, in addition, the most recent observed return z_{t-1}. He considers that "this extra data embodies information about previous regimes and is thus useful".

In [Gra96] and [Kla02], the path dependency problem is apparently solved, but at the price of questionable and rather ad-hoc modifications of Equation (5.39). These modifications may not be justified by economic and/or financial reasonings, or intuitive modeling intuition. They should rather be seen as some technical tricks to avoid the numerical difficulty of maximum likelihood inference. This justifies the introduction of the more realistic and more intuitive Markov-Switching GARCH model of [HMP04] that we study and extend in this paper.

7.3 Technical lemma

Recall the $(p_k + q_k) \times (p_k + q_k)$ random matrix $\mathbf{A}_{k,t}$ defined in Equation (5.19) and its expectation $\mathbf{A}_k$.

time $t-2$, and so on. Consequently, the conditional variance at time t depends on the entire sequence of regimes up to time t. The likelihood function is constructed by integrating over all possible paths".

Lemma 1. *For any* $\lambda \in \mathbb{C}$,

$$det(\lambda I_{p_k+q_k} - \mathbf{A}_k) = \lambda^{p_k+q_k}\Big(1 - \sum_{j=1}^{q_k} a_{k,j}\lambda^{-j} - \sum_{j=1}^{p_k} b_{k,j}\lambda^{-j}\Big).$$

Proof. To lighten notations, let us forget the sub-idex "k" everywhere. Thus, we will prove the result for a $(p+q)\times(p+q)$ matrix now denoted $\mathbf{A}$.

Denote the characteristic polynomial of $\mathbf{A}$ as $D(a_{1:q}; b_{1:p}) := \det(\mathbf{A} - \lambda I_{p+q})$, that is written

$$D(a_{1:q}; b_{1:p}) = \begin{vmatrix} a_1-\lambda & a_2 & a_3 & \cdots & a_q & b_1 & b_2 & & \cdots & & b_p \\ 1 & -\lambda & 0 & \cdots & 0 & 0 & 0 & & \cdots & & 0 \\ 0 & 1 & -\lambda & \ddots & \vdots & 0 & 0 & & \cdots & & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 & \vdots & \vdots & & \cdots & & \vdots \\ 0 & \cdots & 0 & 1 & -\lambda & 0 & 0 & & \cdots & & 0 \\ a_1 & a_2 & \cdots & & a_q & b_1-\lambda & b_2 & & \cdots & & b_p \\ 0 & & \cdots & & 0 & 1 & -\lambda & 0 & \cdots & \cdots & 0 \\ 0 & & \cdots & & 0 & 0 & 1 & -\lambda & 0 & \cdots & 0 \\ \vdots & & \cdots & & \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & & \cdots & & 0 & 0 & \cdots & 0 & 1 & -\lambda & 0 \\ 0 & & \cdots & & 0 & 0 & \cdots & \cdots & 0 & 1 & -\lambda \end{vmatrix}. \tag{5.42}$$

By developing $D(a_{1:q}; b_{1:p})$ with respect to its last column and with obvious notations, we obtain

$$D(a_{1:q}; b_{1:p}) = (-\lambda)D(a_{1:q}; b_{1:p-1}) + (-1)^{p+1}b_pC(a_{1:q}) + (-1)^{p+q+1}b_pE(a_{1:q}), \tag{5.43}$$

where $C(a_{1:q})$ and $E(a_{1:q})$ are the determinants of some companion matrices. To be specific,

$$C(a_{1:q}) := \begin{vmatrix} a_1-\lambda & a_2 & a_3 & \cdots & a_q \\ 1 & -\lambda & 0 & \cdots & 0 \\ 0 & 1 & -\lambda & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & -\lambda \end{vmatrix}, \text{ and}$$

$$E(a_{1:q}) := \begin{vmatrix} 1 & -\lambda & 0 & \cdots & \cdots & 0 \\ 0 & 1 & -\lambda & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \cdots & 0 & 1 & -\lambda & 0 \\ 0 & \cdots & \cdots & 0 & 1 & -\lambda \\ a_1 & a_2 & \cdots & \cdots & a_{q-1} & a_q. \end{vmatrix}.$$

In $C(a_{1:q})$, if we multiply the columns $1, 2, \ldots, q-1$ (from left to right) by $\lambda^{q-1}, \lambda^{q-2}, \ldots, \lambda$

successively, add them to the last column, all elements of the latter column become zero except a single one (located at the upper right corner of the determinant), that equals $a_q + \lambda a_{q-1} + \ldots + \lambda^{q-1} a_1 - \lambda^q$. Since the cofactor of this element is $(-1)^{q+1}$, deduce

$$C(a_{1:q}) = (-1)^{q+1} \{ \sum_{j=0}^{q-1} \lambda^j a_{q-j} - \lambda^q \}. \tag{5.44}$$

Similarly, the same calculation for $E(a_{1:q})$ yields

$$E(a_{1:q}) = \sum_{j=0}^{q-1} \lambda^j a_{q-j}. \tag{5.45}$$

Note that $C(a_{1:q}) + (-1)^q E(a_{1:q}) = (-1)^q \lambda^q$. Recalling and iterating $p-1$ times the relationship (5.42), this provides

$$\begin{aligned} D(a_{1:q}; b_{1:p}) &= (-\lambda) D(a_{1:q}; b_{1:p-1}) + (-1)^{p+q+1} \lambda^q b_p \\ &= (-\lambda)^2 D(a_{1:q}; b_{1:p-2}) + (-1)^{p+q+1} \lambda^{q+1} b_{p-1} + (-1)^{p+q+1} \lambda^q b_p \\ &= \cdots = (-\lambda)^{p-1} D(a_{1:q}; b_1) + (-1)^{p+q+1} \sum_{j=0}^{p-2} \lambda^{q+j} b_{p-j}. \end{aligned} \tag{5.46}$$

By developing $D(a_{1:q}; b_1)$ with respect to its last column, we get

$$D(a_{1:q}; b_1) = (b_1 - \lambda) C(a_{1:q}) + (-1)^q b_1 E(a_{1:q}).$$

Using (5.44) and (5.45), we have

$$(-1)^{q+1} D(a_{1:q}; b_1) = \lambda^{q+1} - \sum_{j=0}^{q-1} \lambda^{j+1} a_{q-j} - \lambda^q b_1.$$

Putting the latter equation into (5.46) yields

$$\begin{aligned} (-1)^{p+q} D(a_{1:q}; b_{1:p}) &= \lambda^{p+q} - \sum_{j=0}^{q-1} \lambda^{p+j} a_{q-j} - \sum_{j=0}^{p-1} \lambda^{q+j} b_{p-j} \\ &= \lambda^{p+q} - \sum_{j=1}^{q} \lambda^{p+q-j} a_j - \sum_{j=0}^{p-1} \lambda^{p+q-j} b_j, \end{aligned}$$

after re-indexing, and yielding the announced result. □

Part III

Learning From Path Signatures

Chapter 6

Clustering Digital Assets Using Path Signatures: Application to Portfolio Construction

Abstract

We propose a new way of building portfolios of cryptocurrencies that provide good diversification properties to investors. First, we seek to filter these digital assets by creating some clusters based on their path signature. The goal is to identify similar patterns in the behavior of these highly volatile assets. Once such clusters have been built, we propose "optimal" portfolios by comparing the performances of such portfolios to a universe of unfiltered digital assets. Our intuition is that clustering based on path signatures will make it easier to capture the main trends and features of a group of cryptocurrencies, and allow parsimonious portfolios that reduce excessive transaction fees. Empirically, our assumptions seem to be satisfied.

Contents

1 Introduction

Optimal portfolio construction research has driven the asset management industry for decades. Many innovative approaches have been proposed. Among all these approaches, mean-variance portfolio and maximum diversification strategies stand out for their intuitive appeal and historical effectiveness. During the last decades, new asset classes emerged, in particular digital assets. These new assets differ from other asset classes by their very high volatility and peculiar dynamics (sequences of booms and bursts, for instance). The high volatility of digital assets is due to several factors probably linked to their youth, including regulatory uncertainty, speculation, and the low liquidity in these new markets. Several researchers (Bukovina, Marticek, et al. [BM+16], Baur and Dimpfl [BD17], Pichl and Kaizoji [PK17], and Inzirillo and De Villelongue [ID22b], etc.) have been studying the behavior and modeling of latent processes to determine the inherent risks of crypto-assets. Other researchers (Balcilar et al. [Bal+17], e.g.) have tried to introduce covariates to predict the performance or the volatility of digital assets. More recently, Schnoering and Inzirillo [SI22] studied synchronous and asynchronous relationships using lead-lag graphs to detect some unexplained dependencies between digital assets. Nonetheless, no consensus has emerged in terms of a reliable model for predicting the price of these new assets, particularly in a multivariate setting. Indeed, the factors that influence their price dynamics are numerous and complicated to understand. Considering many of them simultaneously seems to be a very difficult task.

For these reasons, we will not try to explain/model the price dynamics of crypto assets strictly speaking, for instance with factors and/or covariates as in financial econometrics. We will rather focus on grouping the numerous existing crypto assets into homogeneous classes. By selecting a representative asset in each class, we build a "parsimonious" portfolio of crypto assets that could behave better than other more "naive" or standard portfolios. Indeed, correlation-based clustering has been used to deduce hierarchical connections among diverse asset classes by analyzing the correlation matrix of their returns (Bonanno, Lillo, and Mantegna [BLM01], Song, Chang, and Song [SCS19], Bonanno et al. [Bon+03], and Mantegna [Man99]), with promising results. The contribution of this research lies in its innovative approach to portfolio construction, leveraging path signatures to enhance traditional methodologies. We explore how clustering based on path signatures can refine the asset selection process for mean-variance Markowitz and Todd [MT00] and maximum diversification Choueifaty and Coignard [CC08] portfolios, potentially leading to superior risk-adjusted returns. Here, we will create homogenous clusters on the basis of path signatures, enabling the grouping of digital assets according to their risks and return profiles. The summary of information provided by signatures will enable portfolio managers to process and manage relatively few assets (compared to a naive mean-variance approach, e.g.). Through empirical analysis and computational experiments, we demonstrate the effectiveness of our methodology.

The first step is to estimate path signatures and to create "agnostic" features solely based on historical prices. Path signatures [Che58] are sequences of numbers that describe curves, here some price sequences over fixed periods of time. They are directly calculated from observed

prices. They capture and summarize some empirical characteristics as the slopes and shapes of price curves, in addition to dependencies between successive quotes. The second step involves defining clusters based on these signatures to create a universe of investable digital assets, where each cluster represents a group of assets with "similar" path signatures. Clustering techniques aim to group similar objects according to their characteristics. Such procedures are widely used in various fields and constitute a classical research domain in statistics and machine learning. Here, the objective of clustering is to identify similar price patterns, structures, or relationships among sets of crypto currencies. The different steps of the methodology will be detailed in the next sections. Note that the idea of applying clustering in finance to build optimized portfolios, i.e. to find the right trade-off between diversification and the cost of dynamically managing a large number of assets, is not new: see León et al. [Leó+17] and Korzeniewski [Kor18] or Marvin [Mar15].

Even if the goal is to manage a reduced number of crypto assets in a portfolio, we still hope to benefit from some diversification effects (inside the universe of digital assets). Indeed, when it deals with portfolio diversification many advantages come to our mind:

Risk reduction: this is the main argument in favor of diversification. By spreading investments across different assets that are not perfectly correlated, we can reduce the overall risk of the portfolio. If one investment performs poorly, the others can compensate. Digital assets, because of their youth and their nature, behave very differently from more standard assets. Generally speaking, it is fruitful to further diversify portfolios by integrating assets whose behavior is as far as possible unrelated to the market.

Return potential: by adding numerous assets into a portfolio, there is a hope of including specific assets that are likely to outperform the others at a given time.

Access to broad investment opportunities: a diversified portfolio typically includes equities, bonds, real estate assets, commodities, etc., providing access to a wide range of investment opportunities (Rugman [Rug76]). Despite the relatively low correlation of the digital asset market with traditional markets, its excess volatility is tarnishing its ability to reduce risk and is even becoming a performance driver Baur, Hoang, and Hossain [BHH22]. Path signatures give us a different representation of these time series to identify the relationships or common behaviors between digital assets.

2 Path Signatures

Introduced by Chen [Che58], path signatures are sequences of iterated integrals of (transformed) time series (here, series of digital asset prices). For a detailed presentation of path signatures as a reliable representation or a set of characteristics for unparameterized paths, we refer the reader to Lyons [Lyo14], Lyons [Lyo98], and Chevyrev and Kormilitzin [CK16]. A path signature, sometimes simply called a signature, is actually a mathematical representation that "succinctly" summarizes the pattern of a path. Although derived from rough path theory and stochastic analysis, it has been recently adopted in many other fields, as in machine learning

(Chevyrev and Kormilitzin [CK16], Perez Arribas et al. [Per+18], and Fermanian [Fer21]), time series analysis (Gyurkó et al. [Gyu+13] and Dyer, Cannon, and Schmon [DCS21]), computer vision (Yang et al. [Yan+22] and Li, Zhang, and Jin [LZJ17]), etc.

2.0.1 Definition

Let us define an N-dimensional path $(X_t)_{t\in[0,T]}$. In other words, $X_t = (X_t^1, ..., X_t^N)$. The 1-dimensional coordinate paths will be denoted as (X_t^n), $n \in \{1, \dots, N\}$. To iteratively build the path signature of (X_t), we first consider the increments of $X^1, \dots, X^N$ over any interval $[0, t]$, $t \in [0, T]$, which are denoted $S(X)_{0,t}^1, \dots, S(X)_{0,t}^N$ and defined as follows:

$$S(X)_{0,t}^n := \int_0^t dX_s^n. \tag{6.1}$$

$S(X)_{0,t}^n$ is the first stage to calculate the signature of an unidimensional path. It is a sequence of real numbers, each of these numbers corresponding to an iterated integral of the path. Then, the next signature coefficients will involve two paths: a coordinate path (X_t^m) and the "increment path $(S(X)_{0,t}^n)$ associated to the coordinate path (X_t^n). There are N^2 such second order integrals, which are denoted $S(X)_{0,t}^{1,1} \dots, S(X)_{0,t}^{N,N}$, where

$$S(X)_{0,t}^{n,m} := \int_0^t S(X)_{0,s}^n \, dX_s^m, \ n, m \in \{1, \dots, N\}. \tag{6.2}$$

2.0.2 k-level path signatures

The set of first (resp. second) order integrals involves N (resp. N^2) integrals. The latter first and second order integrals are called the first and second levels of path signatures respectively. Iteratively, we obtain N^k integrals of order k, which is denoted $S(X)_{0,t}^{i_1,\dots,i_k}$ for the k-th level of path signatures, when $i_j \in \{1, \dots, N\}$ and $j \in \{1, \dots, k\}$. More precisely, the k-th level signatures can be written as follows:

$$S(X)_{0,t}^{i_1,\dots,i_k} := \int_0^t S(X)_{0,s}^{i_1,\dots,i_{k-1}} \, dX_s^{i_k}.$$

The path signature $S(X)_{0,T}$ is finally the infinite ordered set of such terms when considering all levels $k \geq 1$ and the path on the whole interval $[0, T]$:

$$\begin{aligned} S(X)_{0,T} := (1, S(X)_{0,T}^1, S(X)_{0,T}^2, \dots, S(X)_{0,T}^N, S(X)_{0,T}^{1,1}, \\ S(X)_{0,T}^{1,2}, \dots, S(X)_{0,T}^{N,N}, S(X)_{0,T}^{1,1,1}, \dots). \end{aligned} \tag{6.3}$$

2.1 Properties of path signatures

Uniqueness

Obviously, there exists a single signature path associated to $(X_t)_{t\in[0,T]}$. The uniqueness of a path signature lies in its ability to compactly and efficiently encode path's information. It means that a path signature can exhaustively capture all "features" of a path, including its geometry,

the directions it tends to follow, the "speed" at which it moves, etc. The uniqueness property of signatures is explored in greater detail in Hambly and Lyons [HL10] and Boedihardjo et al. [Boe+16].

Translation invariance

The value of the path integral $\int_a^b Y_t \, dX_t$ is invariant to X-translation. In other words, it does not change if we shift the entire path (X_t):

$$\int_a^b Y_t \, dX_t = \int_a^b Y_t \, dZ_t, \tag{6.4}$$

where $Z_t = X_t + c$ for every t, for some constant c.

Reparametrisation invariance

Let us define $\psi : [a, b] \to [a, b]$ a function that is onto, continuous, and monotonically non-decreasing. The latter map is termed a "reparametrization". Let us consider two paths $X, Y : [a, b] \to \mathbb{R}$, and let $\psi : [a, b] \to [a, b]$ be a reparametrization of these paths. We can construct new paths $\tilde{X}, \tilde{Y}$: $[a, b] \to \mathbb{R}$, defined by the expressions $\tilde{X}_t = X_{\psi(t)}$ and $\tilde{Y}_t = Y_{\psi(t)}$ for any t. It is noted that

$$d\tilde{X}_t = \dot{\psi}(t) \, dX_{\psi(t)}, \tag{6.5}$$

which leads to

$$\int_a^b \tilde{Y}_t \, d\tilde{X}_t = \int_a^b Y_{\psi(t)} \dot{\psi}(t) \, dX_{\psi(t)} = \int_a^b Y_u \, dX_u, \tag{6.6}$$

The equivalence of the two integrals is obviously obtained through the change of variable $u := \psi(t)$. This confirms the invariance of path integrals with respect to time reparametrization of the paths involved (Chevyrev and Kormilitzin [CK16] and Lyons [Lyo14]). Let us consider a multi-dimensional path $X : [a, b] \to \mathbb{R}^d$ and a reparametrization $\psi : [a, b] \to [a, b]$. As before, denote by $\tilde{X} : [a, b] \to \mathbb{R}^d$ the reparametrized path $(\tilde{X}_t) = (X_{\psi(t)})$. Since every term of the signature $S(X)_{a,b}^{i_1,\ldots,i_k}$ is defined as an iterated path integral of (X_t), it follows from above that

$$S(\tilde{X})_{a,b}^{i_1,\ldots,i_k} = S(X)_{a,b}^{i_1,\ldots,i_k}, \quad \forall k \geq 0, \, i_1, \ldots, i_k \in \{1, \ldots, d\}. \tag{6.7}$$

That is to say, the signature $S(X)_{a,b}$ remains invariant under time reparametrizations of (X_t).

2.2 Signature of a stream

Even if financial time series are always recorded at discrete intervals in practice (closing times, opening times, etc.), it may be considered that such time series evolve continuously in time. Due to its convenience on the theoretical and practical sides, continuous time models are very often used to represent the dynamics of financial assets. Indeed, dealing with continuous intervals implies the use of data streams, sequences of data points $X := (X_1, ...X_N)$ associated

with a timestamp. In the previous sections, we have seen that the signature $S(X)$ of (X_t) is invariant to time reparameterization. However, in some fields, such as economics and finance, there is a need for time reparameterization of the path (X_t).

Let $(\widehat{X_{t_i}})_{i=0}^{N}$ be a data stream. Each point is associated with a specific timestamp. X_{t_i} is the value of the path at time t_i. The objective is to transform this discrete data stream into a continuous path. This transformation can be achieved through a transformation that "connects" all the data points over time. The time-joined transformation Levin, Lyons, and Ni [LLN13], or the lead-lag transformation, allows us to perform this "connection". [Gyu+13] demonstrated that the terms of the path signature (iterated integrals) quantify different path-dependent attributes. This method does not account for the quadratic variation of the process, which is a very important metric in finance. So they achieved the integration of the quadratic variation using the lead-lag transformation of data streams. Given a stream $(\widehat{X}_{t_i})_{i=0}^{N} \in \mathbb{R}^d$, we define its lead-transformed stream $(\widehat{X}_j^{\text{lead}})_{j=0}^{2N}$ by

$$\widehat{X}_j^{\text{lead}} = \begin{cases} \widehat{X}_{t_i} & \text{if } j = 2i, \\ \widehat{X}_{t_i} & \text{if } j = 2i-1. \end{cases}$$ Moreover, we define its lag-transformed stream $(\widehat{X}_j^{\text{lag}})_{j=0}^{2N}$ by

$$\widehat{X}_j^{\text{lag}} = \begin{cases} \widehat{X}_{t_i} & \text{if } j = 2i, \\ \widehat{X}_{t_i} & \text{if } j = 2i+1. \end{cases}$$

Finally, the lead-lag-transformed stream takes values in $\mathbb{R}^{2d}$ and is defined by the paired stream

$$(\widehat{X}_j^{\text{lead-lag}})_{j=0}^{2N} = (\widehat{X}_j^{\text{lead}}, \widehat{X}_j^{\text{lag}})_{j=0}^{2N}.$$

Note that the axis path X^{lead} corresponding to the lead-transform stream is a time-reparametrization of the axis path X, hence

$$S_{t_0,t_N}(\widehat{X}^{\text{lead}}) = S_{0,2N}(\widehat{X}^{\text{lead}}),$$

and similarly

$$S_{t_0,t_N}(\widehat{X}^{\text{lag}}) = S_{0,2N}(\widehat{X}^{\text{lag}}).$$

Moreover, the (signed) area spanned between the ith element of the lead-transform and the jth element of the lag-transform is equivalent to the quadratic cross-variation of the paths $\widehat{X}^i$ and $\widehat{X}^j$.

3 Data and Methodology

In the first step, we select the largest digital assets in terms of market capitalization. Thus, we are confident that the liquidity of each digital asset is sufficient to replicate the strategy in the real world. Since some recently created digital assets listed on the market may face liquidity issues due to their youth, these digital assets should be excluded from this initial investment

universe. Moreover, the most liquid digital assets suffer from less frequent price big jumps than the others. They will not be subject to significant slippage effects when trades are executed, which helps preserve an acceptable tracking error for asset managers.

After identifying this list of digital assets, the objective is to detect connections and/or similarities between the dynamics followed by different digital assets, to create homogenous clusters based on certain "intrinsic" characteristics of their price dynamics. To perform this task, we rely on the k-means algorithm, which is applied to the path signatures of each of the selected digital assets.

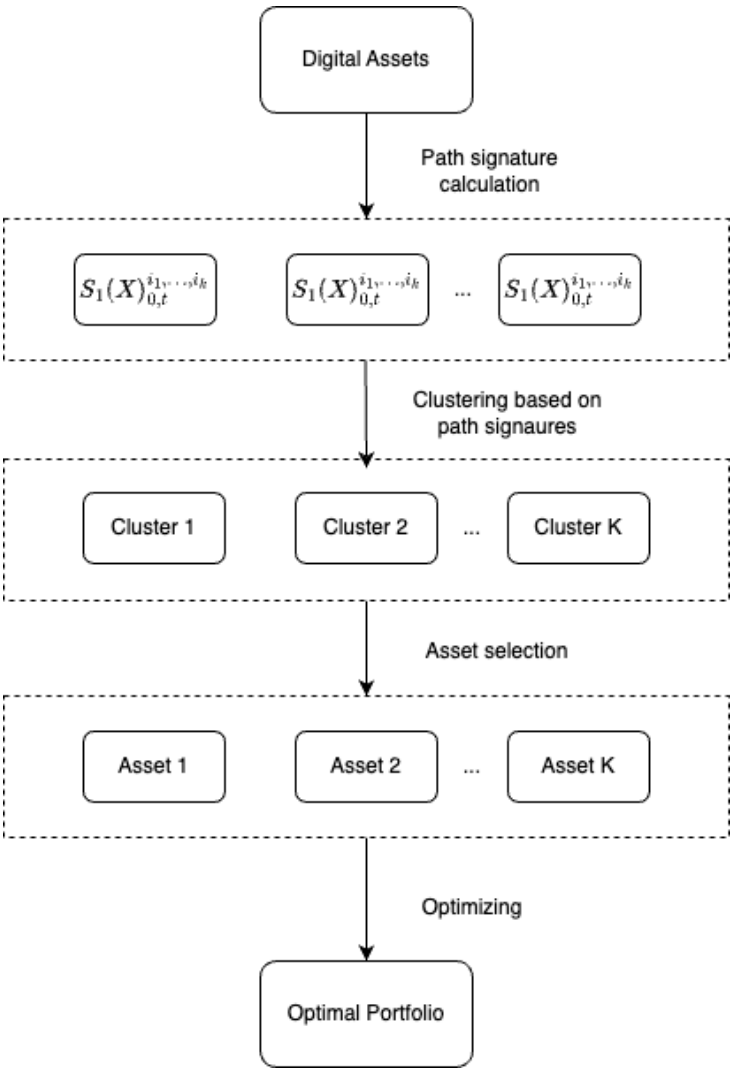


Figure 6.1 – Portfolio construction methodology

Figure 6.1 describes the four steps of our methodology. Our steps are similar to those of Manh and Quoc [MQ23], however, our initial dataset "Digital Assets" will be used as a basis for path signatures calculation, which will be the features of the clustering algorithm. Our portfolio construction flow may be described in four distinct steps. The first step is to calculate the path signatures for each asset within the initial investment universe. The second step consists of grouping digital assets according to their path signatures. During the third step, digital assets

will be selected according to their distance to the centroid of the cluster. For the final step, the optimization algorithm will be applied to the digital assets selected from the third step.

In our experiment, we used a dataset downloaded from Binance. This dataset includes the top 30 digital assets in terms of market capitalisation, according to Coinmarketcap. We downloaded the daily closing price data for all the trading pairs "crypto against USD", and use USDT as a proxy of the US dollar. We only keep the closing prices for each pair, observed at midnight every day. From this data set, we create an initial investment universe at each rebalancing date. In our case, rebalancing events will be set up at midnight every Monday. The rebalancing date is the moment at which the filtered universe is built on top of the initial investment universe. As the number of digital assets tends to grow, some did not yet exist at the start of our backtest. These assets will be added to the investment universe as time goes thanks to our rebalancing mechanism.

For the backtest methodology we will use two types of lookback windows. One is called the Fixed Origin of Time (FOT) method and the other is the traditional rolling window (RW) strategy. The FOT window is growing with time while the rolling window keeps the same size during the whole backtest. The relationships between digital assets evolve over time. We will analyze whether a "static" time frame called the FOT is more reliable than a rolling window, or vice versa, for developing a new portfolio methodology.

Fixed Origin of Time (FOT)

The Fixed Origin of Time (FOT) window methodology is characterized by its progressive expansion over time, enabling the integration of an increasing amount of historical data. The cluster algorithm will incorporate each reshuffle date. We operated on a sequential basis. We downloaded the time series for each asset, focusing solely on their closing prices. Then, we generated a set of dates. We identified the crypto assets that are available at each date, allowing us to construct our investment universe. Then, we gradually expanded our universe of digital assets with the introduction of new ones over time. Let us define N_t as the number of crypto-assets at time t. The vector of path signature Q_t, of the n asset at time t, $n \in [1, ..., N_t]$, is given by

$$Q_t := [S_1(X)_{t_0,t}^{i_1,...,i_k}, S_n(X)_{t_0,t}^{i_1,...,i_k}, ..., S_{N_t}(X)_{t_0,t}^{i_1,...,i_k}], \tag{6.8}$$

Where $S_n(X)_{0,t}^{i_1,...,i_k}$ is the k-th level of path signatures of the n-th digital asset over the interval $[0, t]$. This vector will be the input of the clustering algorithm.

For clustering purpose, we choose the k-means algorithm which aims to divide a set of m points $(x_1, x_2, \ldots, x_m)$ into k groups, each described by the mean of the points in the group. The objective function to be minimized is defined as the sum of the squared distances between each data point and its assigned cluster mean. Given $\{x_i\}_{i=1}^m \in \mathbb{R}^d$, the objective is to identify the centroids $\{c_j\}_{j=1}^k$ of the clusters and to minimize the distance between each data point and

its centroid. The k means loss function is

$$\mathcal{L}(c_1, \Gamma_1, \ldots, c_k, \Gamma_k) := \sum_{j=1}^{k} \sum_{i \in \Gamma_j} d(x_i, c_j), \tag{6.9}$$

where Γ_j denotes the j-th cluster, whose centroid is c_j. We denote $d(x_i, c_j)$ a distance between the data point x_i and the centroid of the j-th cluster. Usually an Euclidean distance is used in this problem, i.e. $d(x_i, c_j) = ||x_i - c_j||^2$.

Let us denote $\{c_j^s\}_{j=1}^k$ our centroids at time t. Once k clusters have been built, we assign each data point to its closest centroid. Here, the "closest" is determined by the Euclidean distance between a point and a centroid. This *assigment step* can be represented as

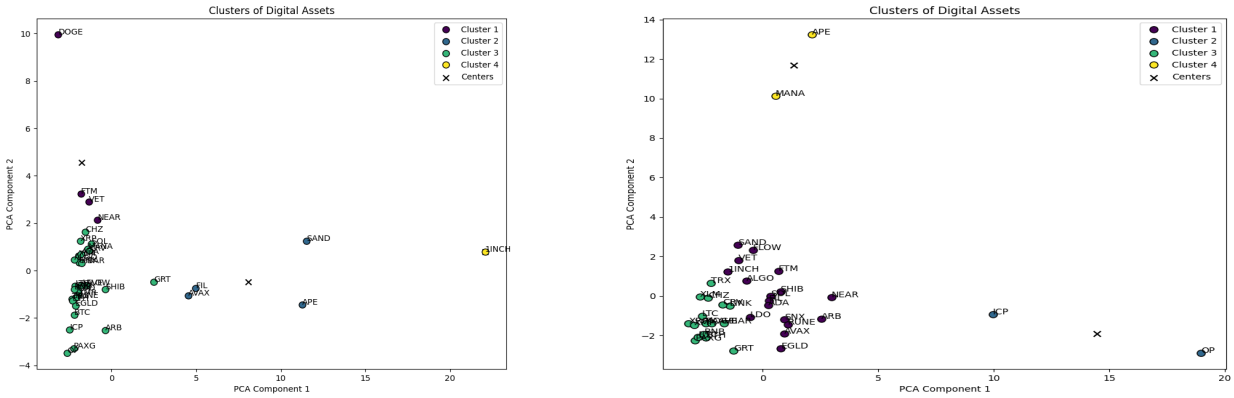
$$\Gamma_j^s := \{i : ||x_i - c_j^s|| \leq ||x_i - c_k^s||, \ \forall k \neq j\}. \tag{6.10}$$

For a given cluster Γ_j^s, the minimizer of (6.9), the centroid of Γ_j is *updated* as follows:

$$c_j^{s+1} = \arg\min_{c_j} \sum_{i \in \Gamma_j^s} ||x_i - c_j||^2 = \frac{1}{|\Gamma_j^s|} \sum_{i \in \Gamma_j^s} x_i. \tag{6.11}$$

The operation is repeated until the convergence of the algorithm when centroids do not change significantly. To illustrate you will find below a picture of the cluster for a given t.

In Figure 6.2a, we plotted the clusters obtained with the FOT methodology at some arbitrarily chosen date. It can be observed that the majority of digital assets including BTC, ETH, and LTC are very close to each other in cluster 2 (in blue), which indicates similarities in terms of path signatures of these digital assets. Digital assets such as SAND, APE and DOGE are located far from the previous points in the chart, which could indicate a more marked difference from the others in terms of characteristics and statistical behavior. 1INCH also differed from the other assets even more strikingly. It seems to have a different behavior. This asset could probably bring an interesting amount of diversification inside a digital asset portfolio. To build clusters using the (FOT) method, we keep a complete history of assets, which is fed continuously as time goes. This means that some assets may not exist at the start, but are added throughout the backtest. This methodology ensures greater stability over time within the clusters. Some unselected digital assets could have been subject to a breach of the exchange platform, a hack or other threats leading to their disappearance. Cluster 2 gathers the largest digital assets in term of market cap. We supposed the latter digital assets move in the save direction “in general” (without any special market condition, such as some news related to a specific digital asset typically). Figure 6.3 gives us an overview of the statistical behavior of the digital assets within the largest cluster (Cluster 1 6.2a). The scatterplots of log returns show that these digital assets are positively dependent. The distributions illustrated by the histograms vary according to each digital assets: we observe histograms with high and sharp peaks which indicates low volatility, and others with wider bases, a sign of commonly observed high volatility. As for the contours, they indicate that for some pairs, there are areas where returns are particularly concentrated, which could signify similar market behaviours.



(a) FOT Clusters

(b) RW Clusters

Figure 6.2 – Comparison of FOT and RW clusters as of 2023-12-25.

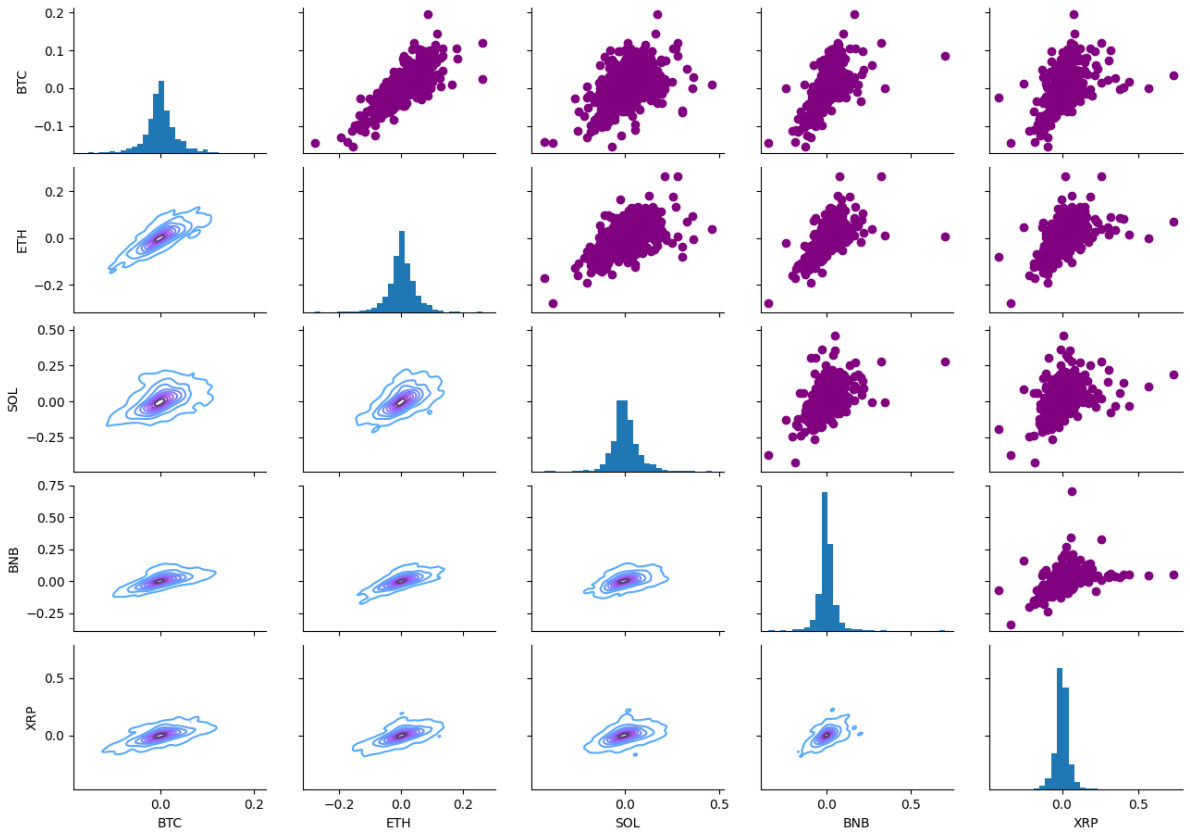


Figure 6.3 – Scatterplots, histograms and joint distributions for the components of Cluster 3 (FOT).

Rolling Window (RW)

The use of the sliding window is particularly relevant in our context of cluster construction at each rebalancing date. At each rebalancement date, the signatures of the existing assets are calculated on the (past) rolling window period of time. The assets that appeared since the last

rebalancement date are then progressively included in the universe of assets. The advantage of this method is its flexibility and its ability to rapidly integrate market changes. By using a sliding window, we assume that recent past price behaviors offer the best indications of future trends. Our objective is to determine whether the rolling window, with its responsiveness to market conditions, provides a more solid insight for portfolio construction than the Fixed Origin of Time window which incorporates a longer historical perspective. This comparison will be made in a latter section, which aims to identify the strategy offering the best balance between taking long-term trends into account and reacting to recent market developments to optimize portfolio performance.

For the Rolling Window (RW) methodology, we maintain a consistent window size of thirty days throughout the backtesting period. Unlike the Fixed Origin of Time (FOT) methodology, which includes more and more historical data (through prices) over time, the rolling window method moves forward in time, keeping the quantity of analyzed data constant but updating it to reflect the most recent price patterns. We define N_t as the number of crypto assets available in our investment universe at time t within the rolling window, and $Q_t, n \in [1, ..., N_t]$ as the vector representing the path signatures of the n-th asset within this window. Q_t is given by:

$$Q_t := [S_1(X)^{i_1,...,i_k}_{t-q,t}, S_n(X)^{i_1,...,i_k}_{t-q,t}, ..., S_{N_t}(X)^{i_1,...,i_k}_{t-q,t}], \tag{6.12}$$

where $S_n(X)^{i_1,...,i_k}_{t-q,t}$ is the path signature of the $n-th$ asset over the time window $[t-q, t]$.

This methodology allows for a dynamic adjustment to our portfolio's asset composition. It ensures that our investment decisions are based on the most recent and relevant data, thus striving to enhance our portfolio's performance by adapting to the most recent trends in the market.

Figure 6.2b show the different clusters using a rolling window of thirty days. Comparing the two methodologies, the RW clusters seem to be more separated, especially for Cluster 2 which is more dispersed. The RW clusters appear to have a tighter central cluster (Cluster 1) and less dispersion in Cluster 2. In both methods, Cluster 3 contains a few assets only. Cluster 4 has got even fewer, and they both appear to have outliers that are far from the cluster centers. The scale on the PCA Component 1 axis is larger for the FOT Clusters than the RW Clusters, indicating a broader distribution of data in the FOT case. Nonetheless, these results are related to a particular day and it is difficult to assess generalities.

4 Application

For our comparative study of digital asset portfolio strategies, we built three different portfolios: equally weighted, mean variance (Markowitz and Todd [MT00]) and maximum diversification (Choueifaty and Coignard [CC08]) portfolios. For our application, our objective is to determine if portfolios refined through clustering techniques can outperform more traditional investment strategies in performance. We started with a portfolio investment universe without imposing any selection criteria. Then, we calculate the portfolio value for each date. We refine

our investment universe by retaining only digital assets nearest to the centroids of each defined class. This approach aims to lower rebalancing costs by selecting only one asset. This approach seeks to reduce portfolio rebalancing costs, which should be lower for portfolios built using clustering than for those on which no filter has been applied. It is expected that portfolios with fewer assets will exhibit relatively higher volatility compared to more diversified ones.

In this section, we compare the performances of the filtered and not-filtered strategies in the case of equally weighted portfolio, the mean-variance portfolio and the maximum diversification portfolio. Our comparisons will be made firstly on the performance of the strategies and secondly on some performance metrics. For each portfolio, we assume we are able to execute rebalancing at closing times (00:00 UTC). The rebalancing frequency is the same for every strategy; the single difference relais on the allocation model that is used and on the upstream clustering that creates a more parsimonious investment universe than was initially planned.

4.1 Equally Weighted Portfolio (EW)

Below, Table 6.1 presents a comparison between the annualized return (resp. annualized volatility) of two portfolios: the equally weighted portfolio (EW) against the equally weighted portfolios with clustering filters (EW_{sc}^{FOT}) or (EW_{sc}^{RW}). (EW_{sc}^{FOT}) has a higher annualized return of 0.9592, indicating better performance and also a lower annualized volatility of 0.6319, which imply it has less risk in terms of the variability of its returns. (EW_{sc}^{RW}) has the highest annualized return of 1.2523, which indicates a more successful strategy in terms of returns, certainly due to the rolling window which during the reshuffle has allowed the selection of some assets that perform better (under a "momentum" perspective).

Table 6.1 – Performance EW Portfolios (FOT)

	Annualized Return	Annualized Volatility
PORTFOLIO_EW	0.5984	0.8219
PORTFOLIO_SIG_CLUSTER_EW_FOT	0.9592	0.6319
PORTFOLIO_SIG_CLUSTER_EW_RW	1.2523	0.7432

Table 6.2 – Risk metrics EW Portfolios (FOT)

	Sharpe	Calmar	MDD
PORTFOLIO_EW	0.7281	0.7291	0.8203
PORTFOLIO_SIG_CLUSTER_EW_FOT	1.5178	1.5879	0.6036
PORTFOLIO_SIG_CLUSTER_EW_RW	1.6849	2.0306	0.6163

Table 6.2, shows some performance metrics of the latter portfolios. For (EW_{sc}^{RW}), we observe a relatively high Sharpe and Calmar ratios (1.6849 and 2.0306, respectively). This indicates a better risk-adjusted return compared to (EW_{sc}^{FOT}) and (EW). Signature-based clustered

stratgies exhibit lower Maximum DrawDown (MDD) than (EW), suggesting that the former strategies have experienced smaller peak-to-trough declines during our studied period, which is desirable in terms of risk management. Globally, (EW_{sc}^{RW}) appears to outperform (EW_{sc}^{FOT}) and (EW) in terms of both return and risk metrics, suggesting that it might be the most efficient option for investors who want to diversify their portfolio, despite the riskness of this asset class. There is also a lower maximum capital loss on the portfolio based on the filtered universe. This is an advantage for decision-making, even if past performance is no guarantee of future performance.

Fixed Origin of Time (FOT)

In this subsection you will find figures of the evolution of the portfolio allocation (weights) as well as the portfolio values backtested with the methodology that uses the Fixed Origin of Time window (FOT)

4.1.1 Without Clustering

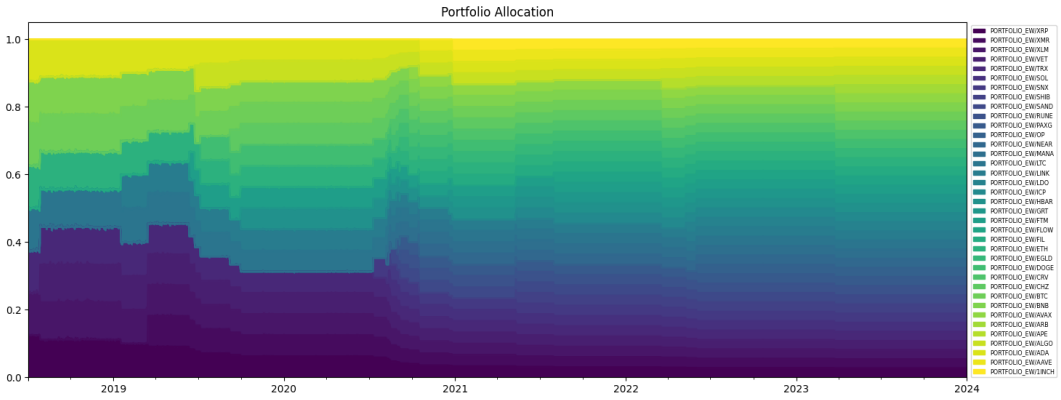


Figure 6.4 – EW Portfolio allocation (FOT)

4.1.2 With Clustering

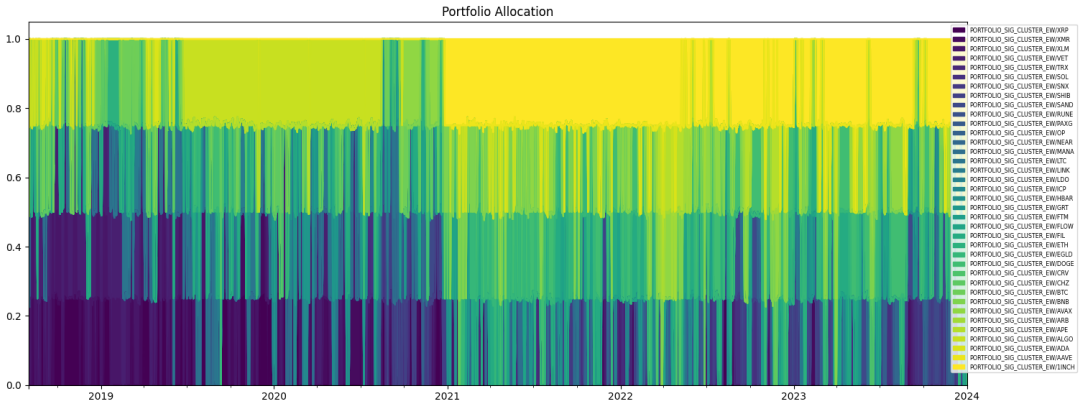


Figure 6.5 – Signature Clustered EW Portfolio allocation (FOT)

Figures 6.4 - 6.5 illustrates the changes in portfolio allocation throughout the backtest period. Specifically, Figure 6.5 perceptibly highlights the introduction of new assets into the portfolio composition over time. Figure 6.6 shows the value of the portfolios over time. We can clearly see that, with annual rebasing, the (EW_{sc}^{FOT}) portfolio is the least volatile.

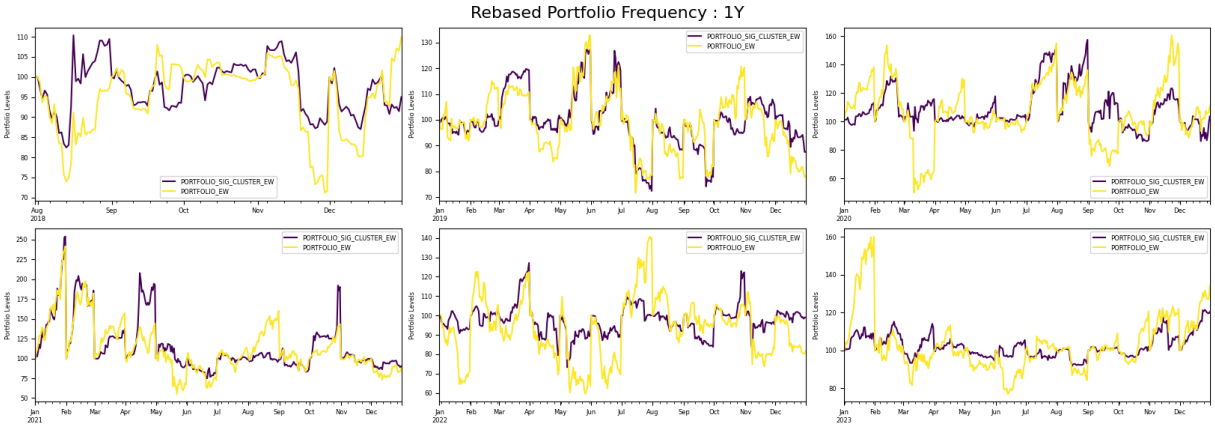


Figure 6.6 – EW Portfolio Rebased Annually (FOT)

Rolling Window (RW)

In this subsection you will find additional figures of the evolution of the portfolio allocation as well as the portfolio values backtested with the methodology that uses the rolling window (RW)

4.1.3 Without Clustering

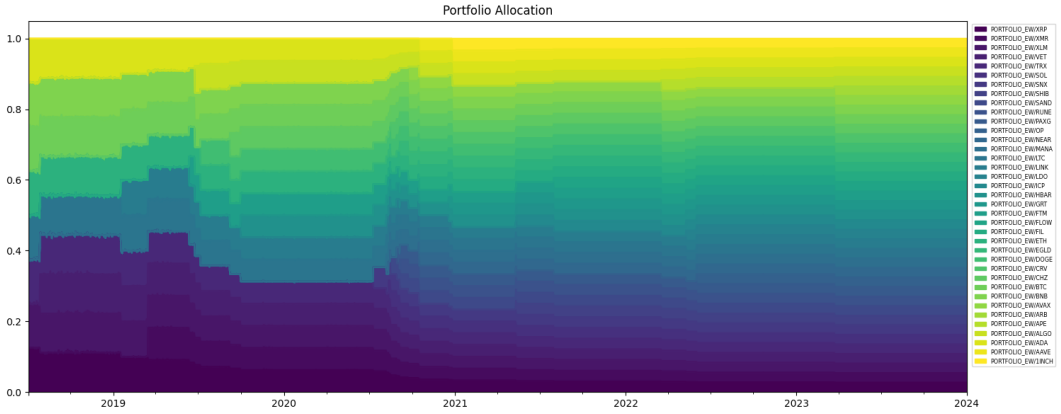


Figure 6.7 – EW Portfolio allocation (RW)

4.1.4 With Clustering

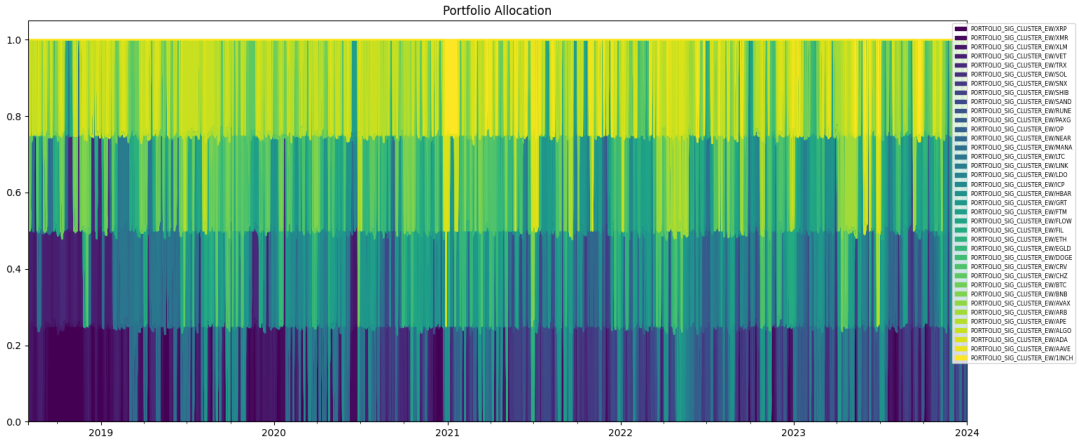


Figure 6.8 – Signature Clustered EW Portfolio allocation (RW)

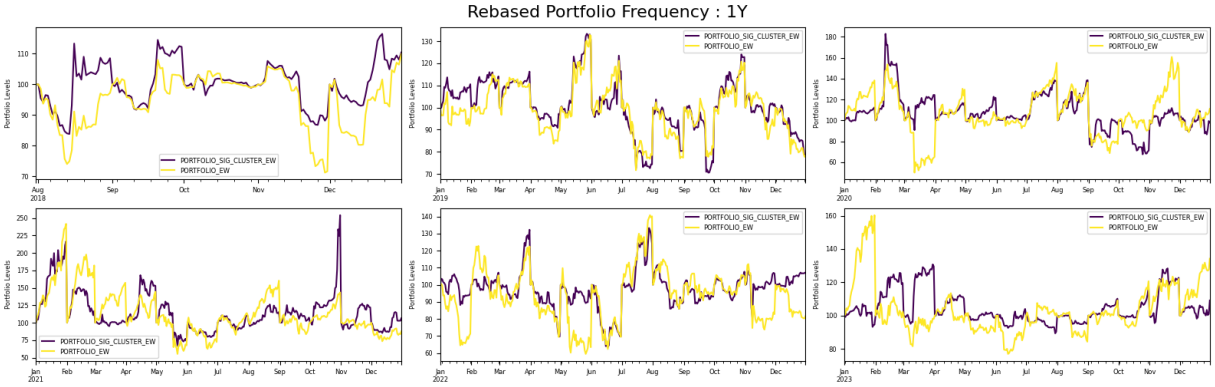


Figure 6.9 – EW Portfolio Rebased Annually (RW)

The next backtesting exercice concerns the MVP strategy, a model where volatility is at the heart of the optimization program. We hope that a reduction of the portfolio size can have a strong impact on the overall portfolio volatility.

4.2 Mininimum Variance Portfolio (MVP)

Table 6.3 compares the annualized return and annualized volatility of two portfolios: the mean variance portfolio (MVP) and mean variance portfolio after clustering filter (MVP_{sc}^{FOT}) or (MVP_{sc}^{RW}). The strategy (MVP_{sc}^{FOT}) shows an annualized return of 0.2199, indicating better performance than the competitiors, but at the price of a higher level of annualized volatility (0.6775 against 0.4262 for (MVP)). When we look at (MVP_{sc}^{RW}), it seems that adapting to market changes and trends using the rolling window is not a good strategy in this case.

Table 6.3 – Performance MV Portfolios

	Annualized Return	Annualized Volatility
PORTFOLIO_MVP	0.1140	0.4262
PORTFOLIO_SIG_CLUSTER_MVP_FOT	0.2199	0.6775
PORTFOLIO_SIG_CLUSTER_MVP_RW	0.1488	0.6675

Table 6.4 – Risk metrics MV Portfolios

	Sharpe	Calmar	MDD
PORTFOLIO_MVP	0.2674	0.2510	0,4539
PORTFOLIO_SIG_CLUSTER_MVP_FOT	0.3245	0.3171	0.6928
PORTFOLIO_SIG_CLUSTER_MVP_RW	0.2229	0.1754	0.8476

Table 6.4 shows the risk metrics for both portfolios, (MVP_{sc}^{FOT}) shows slightly higher Sharpe and Calmar ratios of 0.3245 and 0.3171, respectively. In other words, a better risk-adjusted return compared to (MVP). However, the maximum drawdown of the portfolio with clustering filter is higher than the one with no filter. It seems that the representative asset of the class picked during the portfolio universe construction is catching the most deterministic trend. Figures 6.3-6.4 suggest that (MVP_{sc}^{FOT}) has a better risk-adjusted return than (MVP) portfolio. Considering (MVP_{sc}^{RW}), we might infer that this methodology is not optimal for minimizing the portfolio variance. Utilizing a rolling window approach yields in a lower sharpe ratio of 0.2239, indicating that (MVP_{sc}^{RW}) performs worse compared to (MVP). Therefore, applying a filter on the investment universe in this case may not be effective. For risk management purpose, it is important to note that (MVP_{sc}^{FOT}) remains a more risky portfolio.

Fixed Origin of Time (FOT)

In this subsection you will find figures of the evolution of the portfolio allocation as well as portfolio values backtested with the methodology using the Fixed Origin of Time window (FOT)

4.2.1 Without Clustering

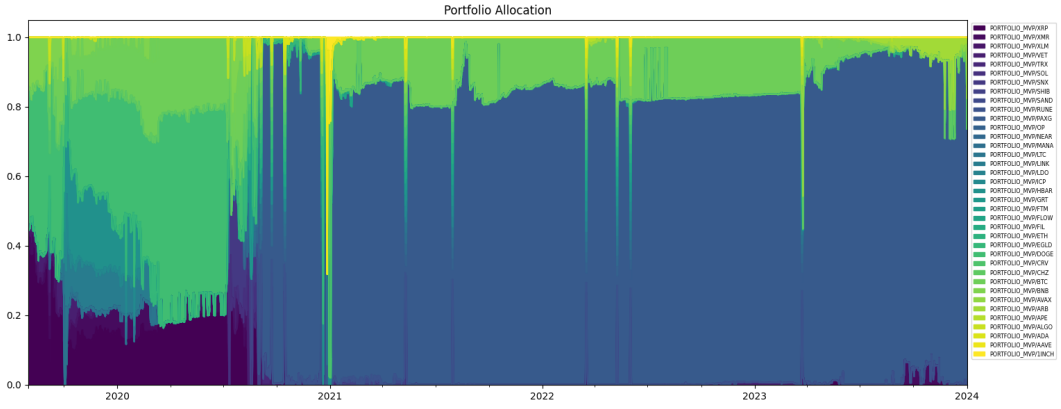


Figure 6.10 – MVP Portfolio allocation (FOT)

4.2.2 With Clustering

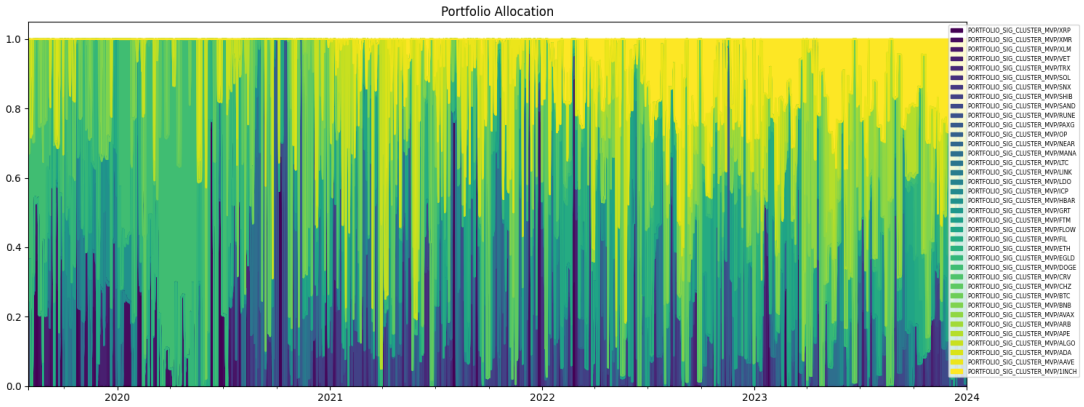


Figure 6.11 – Signature Clustered MVP Portfolio allocation (FOT)

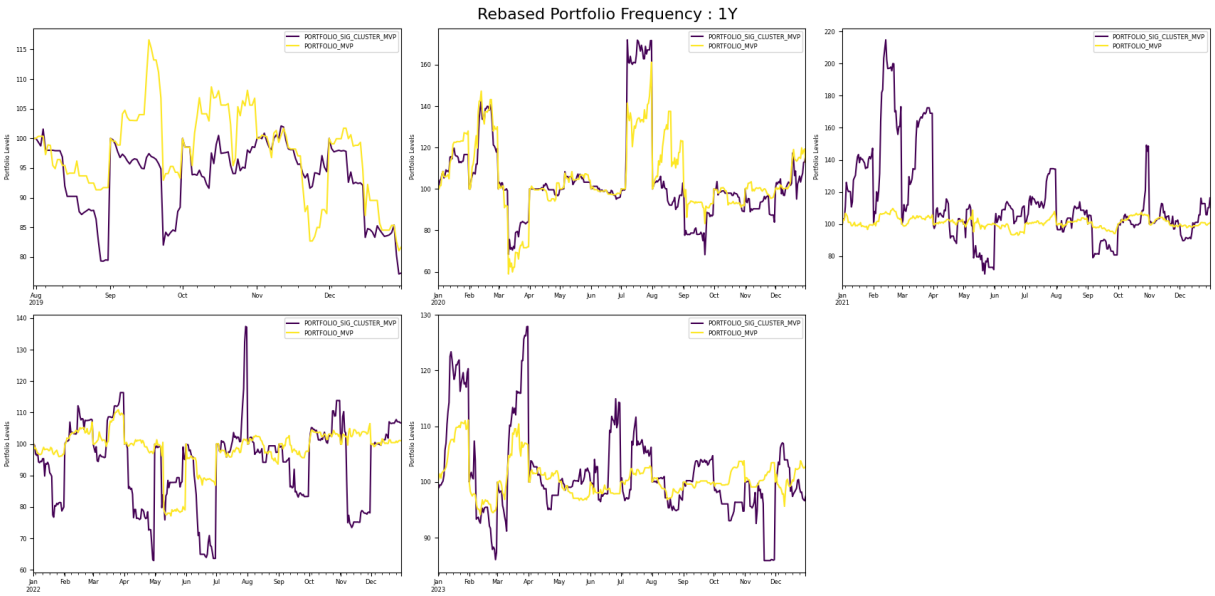


Figure 6.12 – MVP Portfolio Rebased Annually (FOT)

Rolling Window (RW)

In this subsection you will find additional figures of the evolution of the portfolio allocation as well as the value of portfolios backtested with the methodology using the rolling window (RW)

4.2.3 Without Clustering

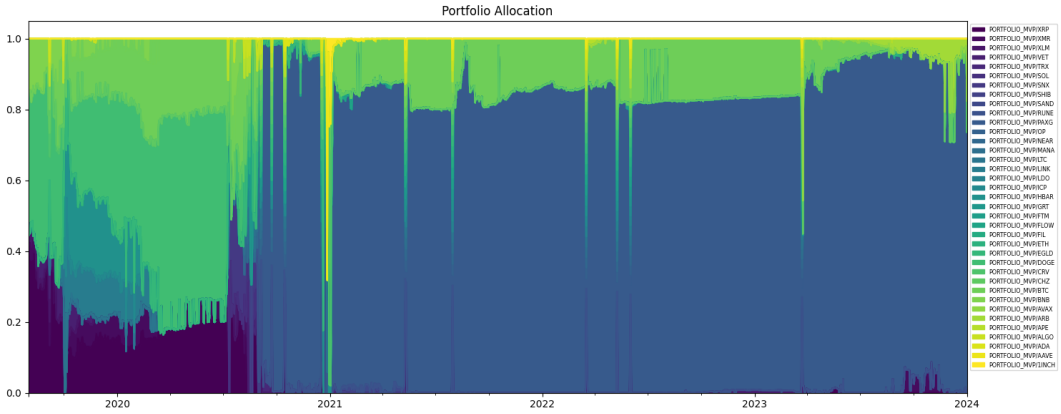


Figure 6.13 – MVP Portfolio allocation (RW)

4.2.4 With Clustering

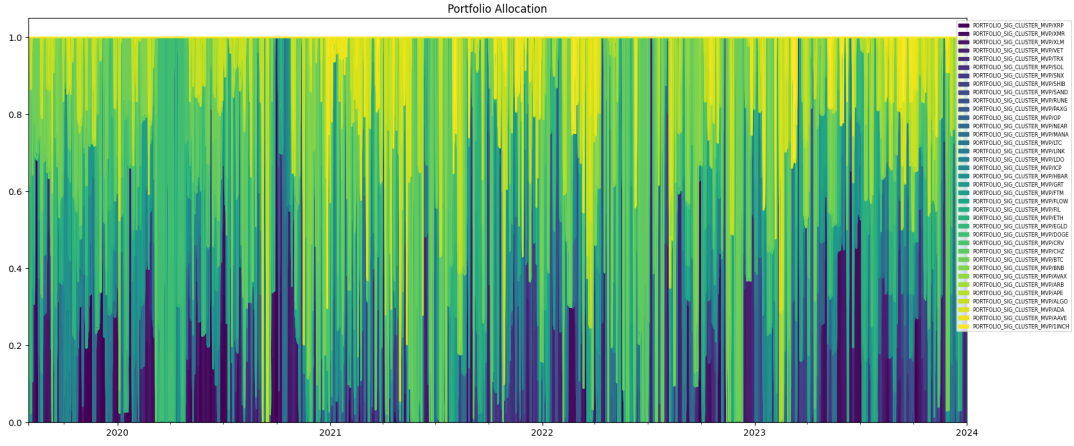


Figure 6.14 – Signature Clustered MVP Portfolio allocation (RW)

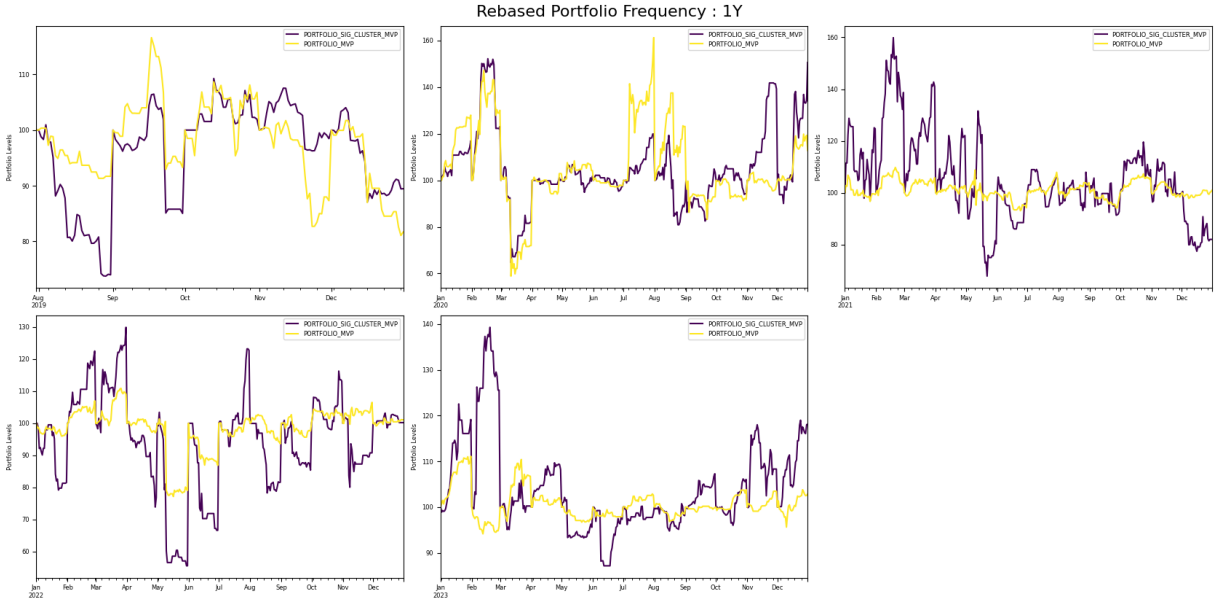


Figure 6.15 – MVP Portfolio Rebased Annually (RW)

4.3 Maximum Diversification Portfolio (MDP)

Table 6.5 shows the comparison of the annualized returns and annualized volatilities of two portfolios - the max diversification portfolio (MDP) and the max diversification portfolio portfolio with clustering filter (MDP_{sc}^{FOT}) and (MDP_{sc}^{RW}). The (MDP_{sc}^{FOT}) shows a higher annualized return of 0.4013. However, it also has a higher level of annualized volatility at 0.6676 compared to 0.5742 for (MDP). If we have a look at (MVP_{sc}^{RW}), it seems that the rolling window (RW) methodology has a really good impact to build the most diversified portoflio. We can clearly see the positive impact on the risk-return trade-off detailed Table 6.6.

Table 6.5 – Performance MDP Portfolios

	Annualized Return	Annualized Volatility
PORTFOLIO_MDP	0.2543	0.5742
PORTFOLIO_SIG_CLUSTER_MDP_FOT	0.4013	0.6676
PORTFOLIO_SIG_CLUSTER_MDP_RW	1.1903	0.7603

Table 6.6 – Risk metrics MDP Portfolios

	Sharpe	Calmar	MDD
PORTFOLIO_MDP	0.4429	0.3974	0.6394
PORTFOLIO_SIG_CLUSTER_MDP_FOT	0.6011	0.5608	0.7151
PORTFOLIO_SIG_CLUSTER_MDP_RW	1.5655	1.6362	0.7268

Table 6.6 shows the risk metrics for both portfolios, (MDP_{sc}^{FOT}) shows slightly higher Sharpe and Calmar ratios of 0.6011 and 0.4429, respectively. This indicates a better risk-adjusted return

compared to (MDP). Looking at the (MDP_{sc}^{RW}) we can clearly see the increase in the sharpe ratio using the RW methodology. There is a performance benefit to using the sliding window methodology, but the maximum drawdown is higher when clustering is used. It seems that the representative asset of the class picked during the portfolio universe construction is catching the most deterministic trend. Overall, both Table 6.5-6.6 suggest that (MDP_{sc}^{RW}) has a better risk-adjusted return than (MDP) and (MDP_{sc}^{FOT}) portfolios. Despite this observation, it is important to note that (MDP_{sc}^{RW}) remains the most risky portfolio.

Fixed Origin of Time (FOT)

In this subsection you will find figures of the evolution of the portfolio allocation as well as the values of portfolios backtested with the methodology using the Fixed Origin of Time window (FOT)

4.3.1 Without Clustering

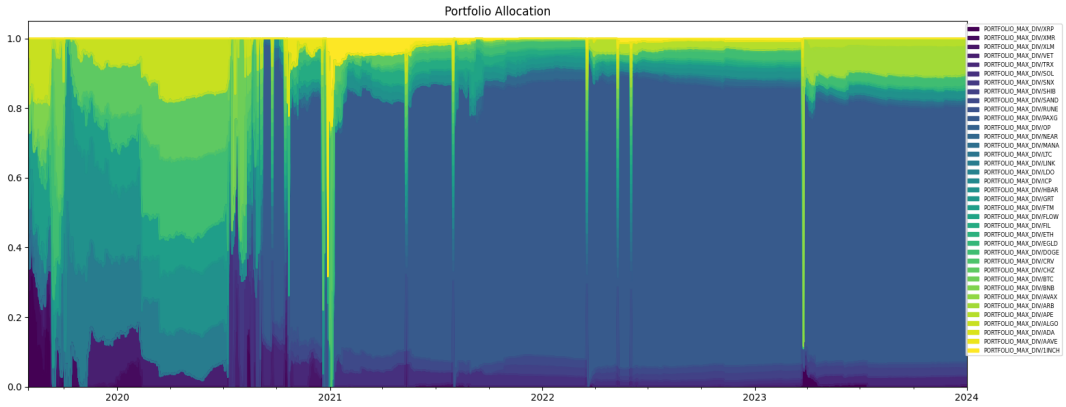


Figure 6.16 – MDP Portfolio allocation (FOT)

4.3.2 With Clustering

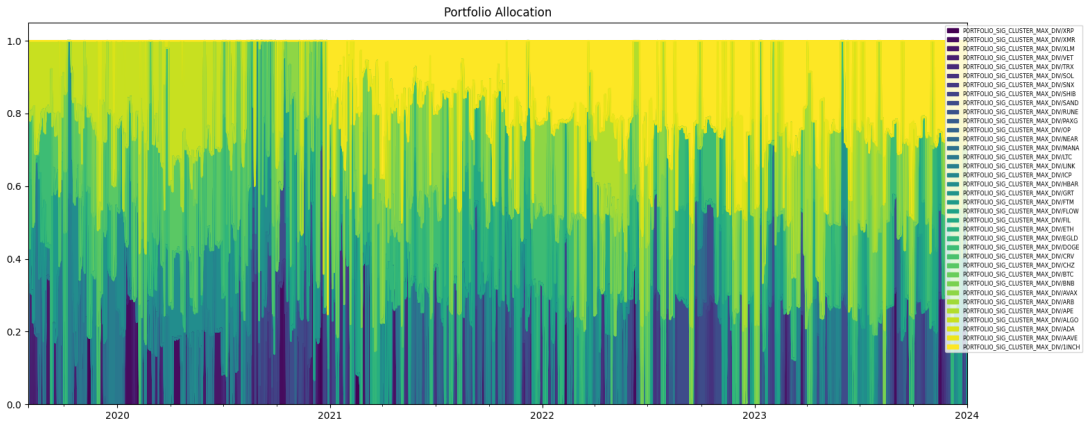


Figure 6.17 – Signature Clustered MDP Portfolio allocation (FOT)

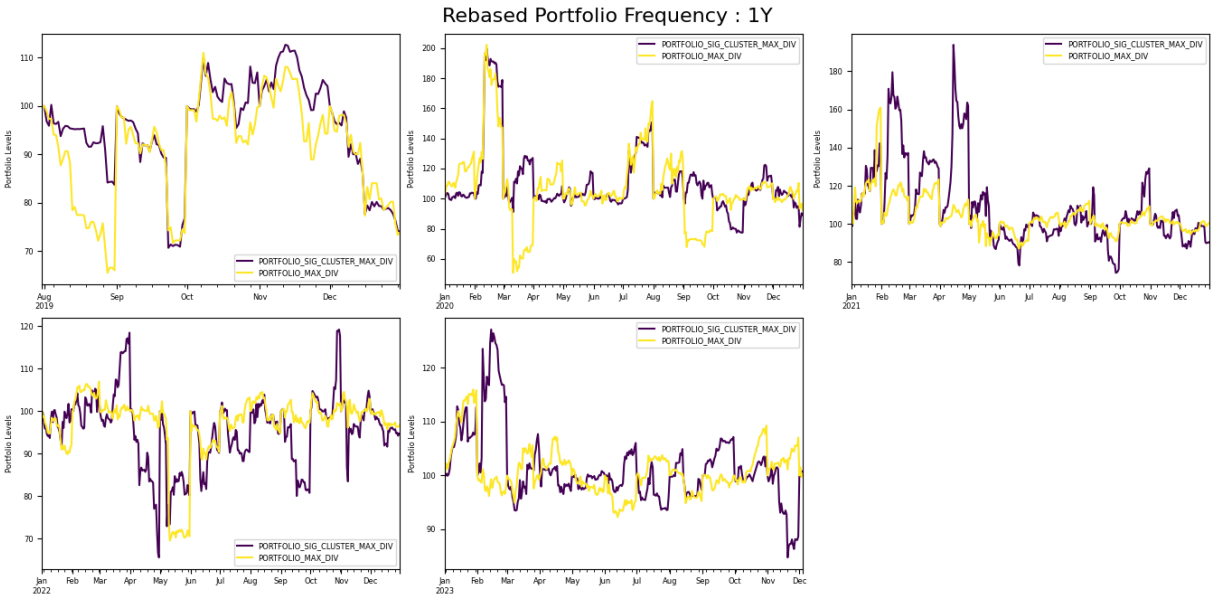


Figure 6.18 – MDP Portfolio Rebased Annually (FOT)

Rolling Window (RW)

In this subsection you will find additional figures of the evolution of the portfolio allocation as well as the values of portfolios backtested with the methodology using the rolling window (RW)

4.3.3 Without Clustering

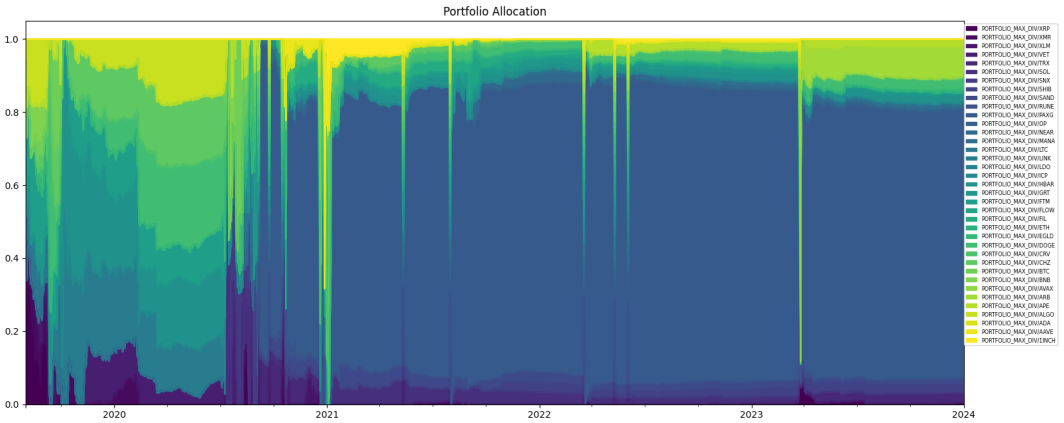


Figure 6.19 – MDP Portfolio allocation (RW)

4.3.4 With Clustering

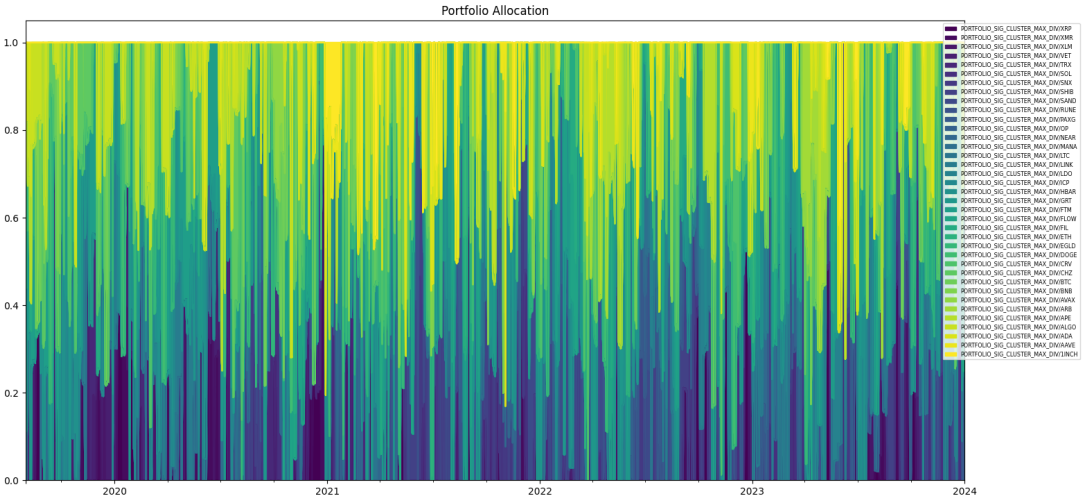


Figure 6.20 – Signature Clustered MVP Portfolio allocation (RW)

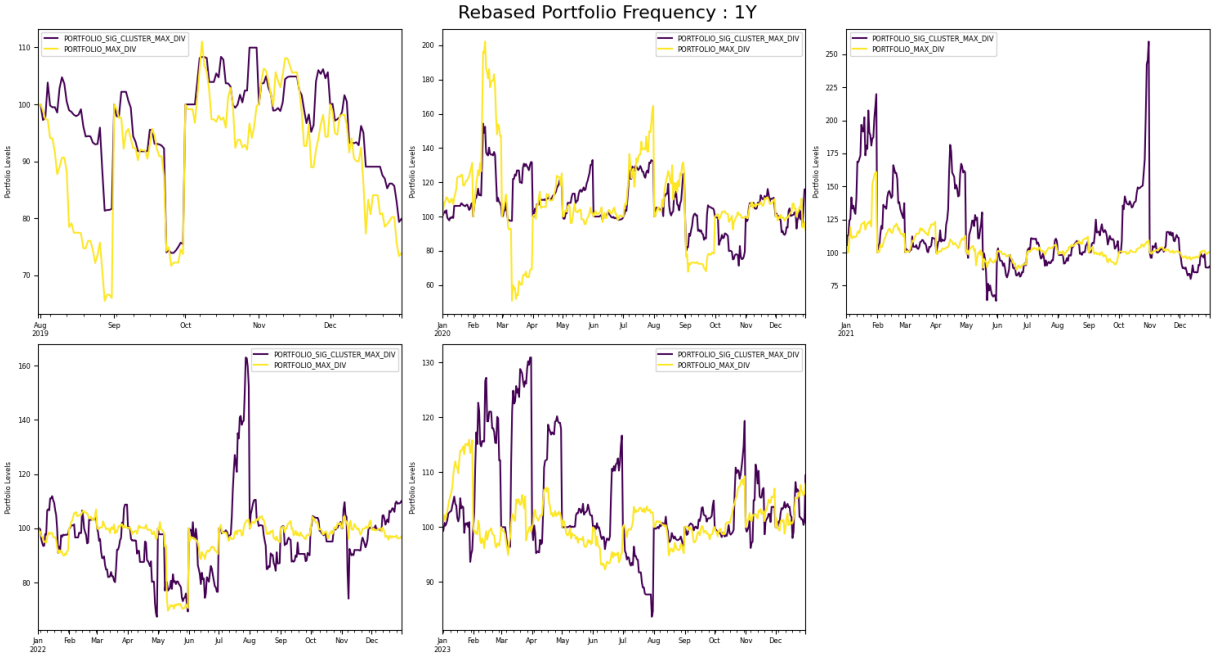


Figure 6.21 – MDP Portfolio Rebased Annually (RW)

4.4 Comparison

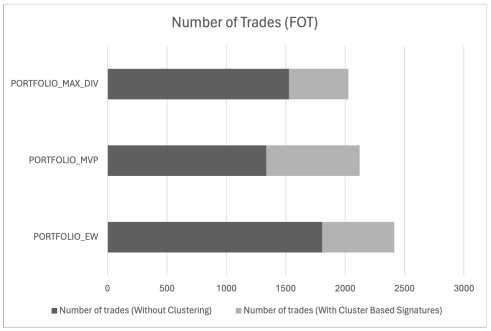


Figure 6.22 – Number of trades (FOT)

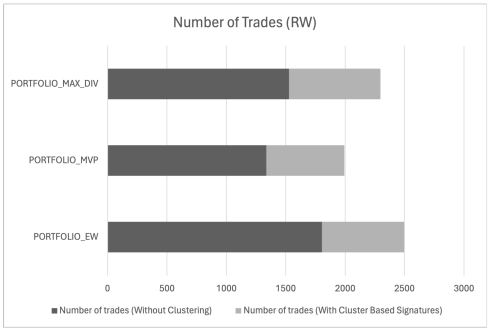


Figure 6.23 – Number of trades (RW)

Figures 6.22-6.23 shows the number of trades for each methodology, using the Fixed of Time (FOT) window and the rolling window (RW). On both figures we can clearly identify the less expensive strategies in term of trading cost. For our backtest we used trading fees of 20 bp. Considering that asset managers may play these strategies we did not considerered a fixed trading fees for each trading, primarily because of the unrealism it suggests.

5 Conclusion

In this chapter, we have introduced a new method to classify financial assets, which differs from traditional approaches centered on historical returns. We have chosen to exploit path signatures, offering a robust alternative for synthesizing the information contained in digital asset prices. This method enables us to discover a different representation, facilitating the identifica-

tion of similarities between various assets. It also allows us to operate in a higher-dimensional space, while taking time dependency into account. The results obtained with different portfolios clearly demonstrate the advantages of this approach, particularly for investors. For the (EW) portfolio, the clustering-filtered version denoted (EW_{sc}^{FOT}) and (EW_{sc}^{RW}) significantly outperforms the standard (unfiltered) methodology in terms of annualized returns (0.9592, 1.2523 vs. 0.5984) and annualized volatility (0.6319, 0.7432 vs. 0.8219). This superior performance is further supported by higher Sharpe and Calmar ratios, and a lower Maximum DrawDown (MDD), indicating a more efficient risk-adjusted return and a lesser decline in value over time. A similar analysis with Mean-Variance portfolios shows that the clustering-filtered MVP (MVP_{sc}^{FOT}) also has a higher annualized return (0.2199 vs. 0.1140) compared to the standard(MVP). However, (MVP_{sc}^{RW}) reports a weak performance of 0.1488 for a level of risk equivalent to the (MVP_{sc}^{FOT}). If we have a look on risk metrics there is a slightly better risk-adjusted returns for (MVP_{sc}^{FOT}), but tempered by a higher MDD. Finally, the (MDP) comparison reveals that the clustering-filtered MDP denoted (MDP_{sc}^{FOT}) and (MDP_{sc}^{RW}) yield a higher annualized return (1.1903, 0.4013 vs. 0.2543) than the standard MDP. The cost of the higher return is the risk of the portfolio, clustering-filtered versions shows an higher annualized volatility (0.7603, 0.6676 vs. 0.5742). However, the risk metrics indicate better risk-adjusted returns for (MDP_{sc}^{RW}), although it experiences a higher MDD. Overall, the clustering-filtered portfolios consistently demonstrate higher returns compared to their standard counterparts, except for the (MVP). However, these benefits are accompanied by increased volatility and MDD, indicating a trade-off between higher returns and increasing risks. Investors should weight these factors carefully, considering their risk tolerance and investment objectives.

Chapter 7

SigKAN: Signature-Weighted Kolmogorov-Arnold Networks (KANs) for Time Series

This part is a joint work with Rémi Genet (Univ. Paris Dauphine).

Abstract

We propose a novel approach that enhances multivariate function approximation using learnable path signatures and Kolmogorov-Arnold networks (KANs). We enhance the learning capabilities of these networks by weighting the values obtained by KANs using learnable path signatures, which capture important geometric features of paths. This combination allows for a more comprehensive and flexible representation of sequential and temporal data. We demonstrate through studies that our SigKANs with learnable path signatures perform better than conventional methods across a range of function approximation challenges. By leveraging path signatures in neural networks, this method offers intriguing opportunities to enhance performance in time series analysis and time series forecasting, among other fields.

Contents

1 Introduction

Forecasting multivariate time series has generated interest from researchers [Wei18; MSW06], and it has grown to be a significant field of study for many businesses. Since there is a growing amount of data available, it makes sense to suggest increasingly sophisticated prediction frameworks. Multivariate time series include multiple interdependent variables, as opposed to univariate time series, which evaluate a single variable over time. The learning task is made considerably more challenging due to these relationships. Complex models that can capture temporal dependencies and dynamic interactions across several dimensions and time horizons are necessary for handling multivariate time series data. Researchers investigated a range of techniques to address the difficulties related to multivariate time series forecasting, such as neural networks [TDF91], state-space models [Ham94; Kim94], vector autoregressive models [ZW06; Lüt13], and neural networks [BMD18; ML19]. Due to its capacity to manage nonlinearities and long-range dependencies in data, deep learning techniques like Long Short-Term Memory (LSTM) networks and attention mechanisms have demonstrated encouraging results [HS97b; Vas+17]. Recently, the Kolmogorov-Arnold Network (KAN) [Liu+24] stands out due to its theoretical foundation in the Kolmogorov-Arnold representation theorem [Kol61] This theorem ensures that any multivariate continuous function can be decomposed into a sum of continuous one-dimensional functions, which is used by KAN to effectively approximate complex functions. Despite their theoretical robustness, there is still much room to improve the expressiveness and efficiency of KANs, especially when processing sequential and temporal data. Even more recently, some research proposed to extend KANs using wavelets [BC24], or proposed new framework to incorporate KAN within RRNs [GI24b]. Some researchers proposed framework to use KANs in time series analysis [Vac+24], and model architecture for time series forecasting using attention [GI24a]. When dealing with sequential and temporal data, the importance of context and feature representation is crucial for accurate and reliable predictions. These highly complex models require a "context" form of representation, summarizing the information in order to improve predictions. Path signatures, originating from rough path theory, offer a powerful method for encoding the essential features of paths or trajectories. These path signatures capture the underlying geometry of the paths through a sequence of iterated integrals, making them particularly well-suited for tasks involving complex sequential data. Integrating path signatures into neural network architectures has shown promise in various applications, but their potential has not been fully explored in the context of KANs. Path signatures, first described by [Che58], are collections of iterated integrals of (transformed) time series, in this case, a series of prices for digital assets. We redirect the reader to [Lyo14; Lyo98; CK16] for a thorough explanation of path signatures as a trustworthy representation or a set of characteristics for unparameterized paths. A path signature, also referred to as a signature for short, is essentially a mathematical expression that "succinctly" captures the structure of a path. While it originated from stochastic analysis and rough path theory, it has recently been applied to a wide range of other fields, including computer vision ([Yan+22; LZJ17]), time series analysis ([Gyu+13; DCS21]), machine learning ([CK16; Per+18; Fer21]), etc. In this paper, we introduce learnable path sig-

natures as a novel improvement to the original KAN layer proposed by Liu et al. [Liu+24]. We introduce a learnable path signature layer that uses learnable parameters to compute path signatures for each input path. The KAN framework then combines these path characteristics with conventional linear transformations. The goal of this integration is to enhance KANs' approximation skills by utilizing the rich geometric information that path signatures provide. Codes are available at SigKAN repository and can be installed using the following command: `pip install sigkan`. Additionally, we release a package needed to compute the path signature using Tensorfow v2.0. iisignature-tensorflow-2 and can be installed using the following command: `pip install iisignature-tensorflow-2`. This package allows signatures to be trainable in custom tensorflow layer, although it does not support GPU utilization.

2 Kolmogorov-Arnold Networks (KANs)

Multi-Layer Perceptrons (MLPs) [HSW89] extension of the original perceptron proposed by [Ros58], are inspired by the universal approximation theorem [Cyb89] which states that a feed-forward network (FFN) with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$. Kolmogorov-Arnold Network (KAN) focuses on the Kolmogorov-Arnold representation theorem [Kol61]. The Kolmogorov-Arnold representation theorem states that any multivariate continuous function can be represented as a composition of univariate functions and the addition operation,

$$f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right), \tag{7.1}$$

where $\phi_{q,p}$ are univariate functions that map each input variable (x_p) such $\phi_{q,p} : [0,1] \to \mathbb{R}$ and $\phi_q : \mathbb{R} \to \mathbb{R}$. Since all functions to be learned are univariate functions, we can parametrize each 1D function as a B-spline curve, with learnable coefficients of local B-spline basis functions. The key insight comes when we see the similarities between MLPs and KAN. In MLPs, a layer includes a linear transformation followed by nonlinear operations, and you can make the network deeper by adding more layers. Let us define what a KAN layer is

$$\mathbf{\Phi} = \{\phi_{q,p}\}, \qquad p = 1, 2, \cdots, n_{\text{in}}, \qquad q = 1, 2 \cdots, n_{\text{out}}, \tag{7.2}$$

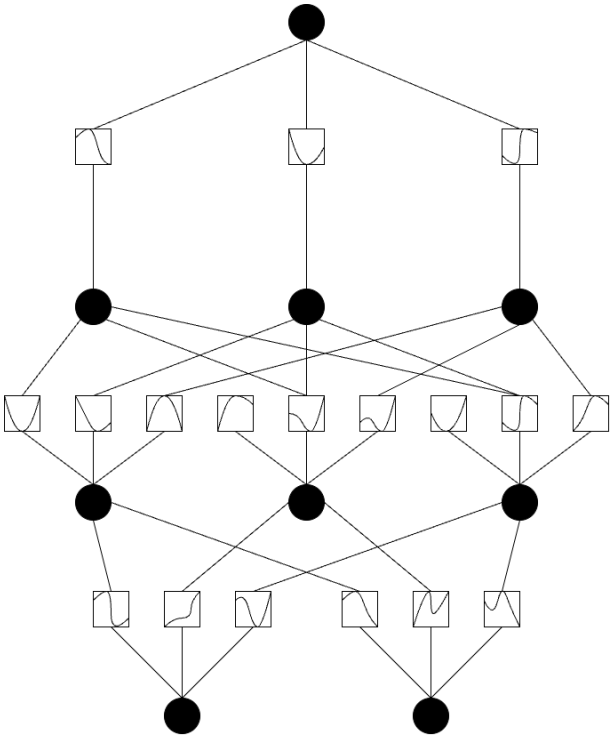


Figure 7.1 – KAN: Kolmogorov Arnold Networks

where $\phi_{q,p}$ are parametrized functions of learnable parameters. In the Kolmogorov-Arnold theorem, the inner functions form a KAN layer with $n_{\text{in}} = n$ and $n_{\text{out}} = 2n + 1$, and the outer functions form a KAN layer with $n_{\text{in}} = 2n + 1$ and $n_{\text{out}} = 1$. So the Kolmogorov-Arnold representations in Eq. (7.1) are simply compositions of two KAN layers. Now it becomes clear what it means to have a Deep Kolmogorov-Arnold representation. Taking the notation from [Liu+24] let us define a shape of KAN $[n_0, n_1, \cdots, n_L]$, where n_i is the number of nodes in the i^{th} layer of the computational graph. We denote the i^{th} neuron in the l^{th} layer by (l, i), and the activation value of the (l, i)-neuron by $x_{l,i}$. Between layer l and layer $l + 1$, there are $n_l n_{l+1}$ activation functions: the activation function that connects (l, i) and $(l + 1, j)$ is denoted by

$$\phi_{l,j,i}, \quad l = 0, \cdots, L-1, \quad i = 1, \cdots, n_l, \quad j = 1, \cdots, n_{l+1}. \tag{7.3}$$

The pre-activation of $\phi_{l,j,i}$ is simply $x_{l,i}$; the post-activation of $\phi_{l,j,i}$ is denoted by $\tilde{x}_{l,j,i} \equiv \phi_{l,j,i}(x_{l,i})$. The activation value of the $(l + 1, j)$ neuron is simply the sum of all incoming post-activations:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,j,i} = \sum_{i=1}^{n_l} \phi_{l,j,i}(x_{l,i}), \qquad j = 1, \cdots, n_{l+1}. \tag{7.4}$$

A general KAN network is a composition of L layers: given an input vector $\mathbf{x}_0 \in \mathbb{R}^{n_0}$, the output of KAN is

$$\text{KAN}(\mathbf{x}) = (\mathbf{\Phi}_{L-1} \circ \mathbf{\Phi}_{L-2} \circ \cdots \circ \mathbf{\Phi}_1 \circ \mathbf{\Phi}_0)\mathbf{x}. \tag{7.5}$$

3 Model Architecture

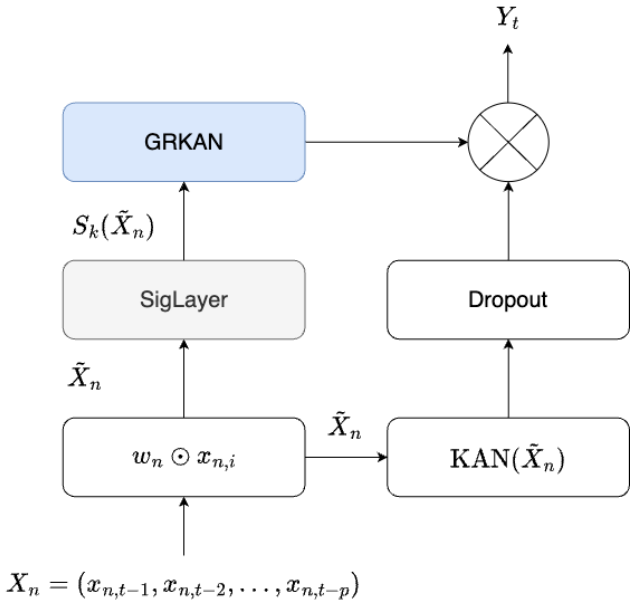


Figure 7.2 – SigKAN

In this section, we will dive into the various components of the model. We will detail each layer and mechanism. We aim to provide a comprehensive understanding of how the model operates and processes data to achieve its predictive capabilities. Before starting, we define the main components of the SigKAN:

— **Gated Residual KAN:** This layer enables the modulation of information flow during the learning task.

— **Learnable path signature layer:** This layer will compute the path signatures for each sequence with learnable parameters.

In simple words, the Signature-Weighted KANs (SigKANs) are composed of SigLayers and KAN layers. As path signatures capture important geometric features of paths, combining the set of information with the output of KAN layers will enhance the power of prediction. Let us define a N-dimensional path $(X_t)_{t\in[0,T]}$ and $X_t = (X_{1,t}, ..., X_{N,t})$. The 1-dimensional coordinate paths will be denoted as $(X_{n,t})$, $n \in \{1, \ldots, N\}$. To build the path signature of (X_t), we first consider the increments of $X^1, \ldots, X^N$ over any interval $[0, t]$, $t \in [0, T]$, which are denoted $S(X)^1_{0,t}, \ldots, S(X)^N_{0,t}$ and defined as follows:

$$S(X)^n_{0,t} = \int_0^t dX^n_s. \tag{7.6}$$

$S(X)^n_{0,t}$ is the first stage to calculate the signature of a unidimensional path. It is a sequence of real numbers, each of these numbers corresponding to an iterated integral of the path. Then, the next signature coefficients will involve two paths: a coordinate path (X^m_t) and the increment path $(S(X)^n_{0,t})$ associated to the coordinate path (X^n_t). There are N^2 such second order integrals, which are denoted $S(X)^{1,1}_{0,t} \ldots, S(X)^{N,N}_{0,t}$, where

$$S(X)^{n,m}_{0,t} = \int_0^t S(X)^n_{0,s}\, dX^m_s,\ n, m \in \{1, \ldots, N\}. \tag{7.7}$$

The set of first (resp. second) order integrals involves N (resp. N^2) integrals. The latter first and second order integrals are called the first and second levels of path signatures respectively. Iteratively, we obtain N^k integrals of order k, which is denoted $S(X)^{i_1,..,i_k}_{0,t}$ for the k-th level of path signatures, when $i_j \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, k\}$. More precisely, the k-th level signatures can be written

$$S(X)^{i_1,\ldots,i_k}_{0,t} = \int_0^t S(X)^{i_1,\ldots,i_{k-1}}_{0,s}\, dX^{i_k}_s.$$

The path signature $S(X)_{0,T}$ is finally the infinite ordered set of such terms when considering all levels $k \geq 1$ and the path on the whole interval $[0, T]$:

$$\begin{aligned} S(X)_{0,T} = (1, S(X)^1_{0,T}, S(X)^2_{0,T}, \ldots, S(X)^N_{0,T}, S(X)^{1,1}_{0,T}, \\ S(X)^{1,2}_{0,T}, \ldots, S(X)^{N,N}_{0,T}, S(X)^{1,1,1}_{0,T}, \ldots). \end{aligned} \tag{7.8}$$

3.1 Gated Residual KANs

The relationship between temporal data is a key issue. Gated Residual Networks (GRNs) offer an efficient and flexible way of modelling complex relationships in data. They allow controlling the flow of information and facilitate the learning tasks. They are particularly useful in areas where nonlinear interactions and long-term dependencies are crucial. In our model, we use the Gated Residual Kolmogorov-Arnold Networks (GRKAN) inspired by the GRN proposed by [Lim+21], we kept the same architecture. We propose a new approach using two KAN layers to control the information flow while bringing more interpretability. Using GRKAN there is no more need for context, which is contained in path signature; however, an additional linear layer is required to match the signature transform and the output of the gating mechanism.

$$\mathrm{GRN}_\omega(x) = \mathrm{LayerNorm}\left(x + \mathrm{GLU}_\omega(\eta_1)\right), \tag{7.9}$$

$$\eta_1 = \mathrm{KAN}(\varphi_{\eta_1}(.), \eta_2), \tag{7.10}$$

$$\eta_2 = \mathrm{KAN}(\varphi_{\eta_2}(.), x). \tag{7.11}$$

In this context, we used activation functions for KAN layers denoted, $\varphi_{\eta_1}(.)$ and $\varphi_{\eta_2}(.)$, SiLU [EUD18] and ELU [CUH15], respectively, while $\eta_1 \in \mathbb{R}^{d_{model}}$ and $\eta_2 \in \mathbb{R}^{d_{model}}$ represent intermediate layers. The standard layer normalization LayerNorm is that described in [BKH16], and ω is an index used to indicate weight sharing. When the expression $\mathrm{KAN}(x)$ is largely positive,

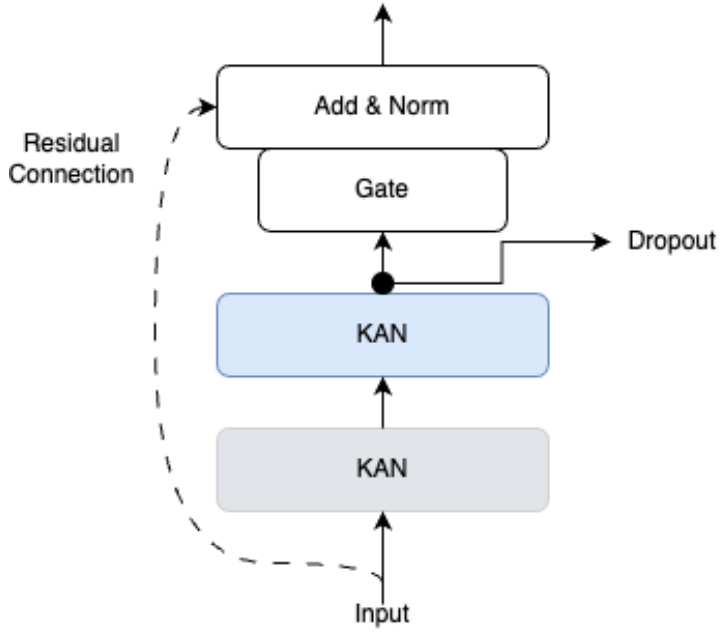


Figure 7.3 – Gated Residual KAN (GRKAN)

ELU activation works as an identity function. On the other hand, when this expression is largely negative, ELU activation produces a constant output, thus behaving like a linear layer. We use Gated Linear Units (GLUs) [Dau+17] to provide the flexibility to suppress any parts of the architecture that are not required for a given dataset. Letting $\gamma \in \mathbb{R}^{d_{model}}$ be the input, the GLU then takes the form:

$$\text{GLU}_\omega(\gamma) = \sigma(W_{4,\omega}\ \gamma + b_{4,\omega}) \odot (W_{5,\omega}\ \gamma + b_{5,\omega}), \tag{7.12}$$

where $\sigma(.)$ is the sigmoid activation function, $W_{(.)} \in \mathbb{R}^{d_{model} \times d_{model}}, b_{(.)} \in \mathbb{R}^{d_{model}}$ are the weights and biases, $\odot$ is the element-wise Hadamard product, and d_{model} is the hidden state size. GLU allows us to control the extent to which the GRN contributes to the original input x – potentially skipping over the layer entirely if necessary as the GLU outputs could be all close to 0 in order to suppress the nonlinear contribution.

3.2 Learnable Path Signature

For a path X_n represented as a sequence of points $(x_{n,1}, x_{n,2}, \ldots, x_{n,N})$, we transform each point with learnable coefficients:

$$\tilde{X}_{n,i} = w_n \odot x_{n,i} \tag{7.13}$$

where w_n is a learnable vector of coefficients and $\odot$ denotes element-wise multiplication. The k-th order signature can be expressed as:

$$S(\tilde{X}) = \left(1, \left(\int_0^1 d\tilde{X}_{t_1}^{i_1}\right)_{1\leq i_1\leq N}, \left(\int_0^1 \int_0^{t_1} d\tilde{X}_{t_2}^{i_2} d\tilde{X}_{t_1}^{i_1}\right)_{1\leq i_1,i_2\leq N}, \ldots, \left(\int_0^1 \int_0^{t_1} \cdots \int_0^{t_{k-1}} d\tilde{X}_{t_k}^{i_k} \cdots d\tilde{X}_{t_2}^{i_2} d\tilde{X}_{t_1}^{i_1}\right)_{1\leq i_1,i_2,\ldots,i_k\leq N}, \ldots\right). \tag{7.14}$$

After performing this step, we will compute the path signature of each sequence of transformed points $(\tilde{X}_1, ..., \tilde{X}_N)$. The transformed path $\tilde{X}_1$ is now a vector of transformed points $(\tilde{x}_{1,1}, \tilde{x}_{1,2}, \ldots, \tilde{x}_{1,N})$. For N paths $(X_1, ..., X_N)$ the path signature transformation

$$S(\tilde{X}) = \left[S(\tilde{X})_1, S(\tilde{X})_2, S(\tilde{X})_3, ...\right], \tag{7.15}$$

The path signature obtained provides a detailed representation of the underlying path, capturing patterns and dependencies accross the data. This can be highly informative for the prediction, and potentially leading to better performance. To do so, the ouput from (7.15) will be the input of a GRKAN 7.3, however it is possible to use another gated mechanism to control the flow of information, enabling the networks to focus on the most relevant features and suppress less useful ones.

$$h_s = f(S_k(\tilde{X})), \tag{7.16}$$

where $f(.)$ is a succession of operations, in our case we choose to use a Gated Residual KAN (GRKAN), the output of this channel will weight the output of the KAN layers.

3.3 Output

We denoted seq and d_{out}, the sequence lenght and the ouput dimension, respectively. The output of the GRKAN is used as a weighting applied on the output of a KAN layer on $(\tilde{X})$. Let us denote the output of the GRKAN as h_s, this one is then passed through a softmax activation

$$\psi = \text{SoftMax}(h_s). \tag{7.17}$$

The global ouput of the SigKAN layer is obtained from

$$\psi \odot \text{KAN}(\tilde{X}). \tag{7.18}$$

It is to note that $\tilde{X}$ is a matrix of shape (seq, d_{out}), while $KAN(\tilde{X}) \in \mathbb{R}^{(seq,d_{out})}$ where the number of output is a parameter given to the layer. The KAN transformation applied is the same on each element of the sequence. On the another hand $\psi \in \mathbb{R}^{d_{out}}$ is a vector obtained by a transformation of the signature in the GRKAN, and thoose weights are applied similarly at each elements of the sequences. The output of the layers still have the sequence dimension, the

SigKAN layers are thus stackable. Therefore, we can generalize a SigKANs network as follows:

$$\begin{aligned} h_0 &= x, \\ h_j &= \mathrm{SoftMax}(\mathrm{GRKAN}_j(S(\tilde{h}_{j-1}))) \odot \mathrm{KAN}_j(h_{j-1}), \quad j = 1, 2, \ldots, L, \\ y &= h_L. \end{aligned} \tag{7.19}$$

The output of the SigKANs still have the sequence dimension, and it would require to either flatten the output, only select the last outputs on this dimension, or apply a RNN that only returns the last hidden state in order to remove this dimension in the network after it. Even if they are stackable, we have observed, in our experiments, that most of the time this is non-desireable as it doesn't yield to increase in performance, as well as increasing the computational cost quickly. The size of the signature depends on the input shape raised to the power of the degree used. This, combined with the lack of GPU support for the signature computations, limits the practical viability of stacking many SigKAN layers, even though this can be beneficial when using standard fully connected layers instead of KAN layers.

4 Learning Task

To illustrate the benefits of our architecture, we performed two different tasks, one being the same as in [GI24b] and [GI24a] for easy comparison, the other being a task where these two tasks are more difficult to perform.

4.1 Task definitions and dataset

The first task we use involves predicting the notional amount traded on the market several times in advance. This is the task used in [GI24b] and [GI24a], and is interesting because volumes exhibit multiple patterns such as seasonality and autocorrelation.The second task changes only the target value; instead of trying to estimate future volume, we aim to predict future absolute return over several time steps in advance. The tasks focus solely on the Binance exchange, so our dataset only contains data from this exchange, which has been the most important market for many years. We have also only used USDT markets, as this is the most widely used stablecoin, and all notionals are therefore intended in USDT.

For the first task, our dataset consists of the notional amounts traded each hour on different assets: BTC, ETH, ADA, XMR, EOS, MATIC, TRX, FTM, BNB, XLM, ENJ, CHZ, BUSD, ATOM, LINK, ETC, XRP, BCH and LTC, which are to be used to predict just one of them, BTC. For the second task, we took the absolute percentage change between closing prices only on BTC to predict BTC. For both, the data period spans 3 years from January 1, 2020 to December 31, 2022.

4.2 Data preprocessing

Data preparation is an important task and can largely alters the quality of the prediction obtained. It is important to have features with the same scale between them, or to have the

same target scale over time. To achieve this for the first task, we use two-stage scaling. The first step is to divide the values in the series by the moving median of the last two weeks. This moving median window is also shifted by the number of steps forward we want to predict, so as not to include foresight. This first preprocessing aims to make the series more stationary over time. The second pre-processing applied is a simple MinMaxScaling per asset, even if here the minimum of the series is 0, it is simply a matter of dividing by the maximum value. The aim is to scale the data in the 0, 1 interval to avoid an explosive effect when learning due to the power exponent. However, this pre-processing is adjusted on the training set, the adjustment being limited to finding the maximum value for each series, which is then used directly on the test set. This means that on the test set, it is possible to have data greater than 1, but as no optimization is used, this is not a problem. For the second task, as there is only one asset and volatility is a more stationary process over time than volumes, we divide the values only by the maximum value of the ream, in order to have values between 0 and 1 only. Finally, we split our data for both set into a training set and a test set, with a standard proportion of 80-20. This represents over 21,000 points in the training set and 5,000 in the test set.

4.3 Loss Function for Model Training

Since we have a numerical prediction problem for both tasks, we have opted to optimize our model using the root mean square error (RMSE) as the loss function, whose formula is simple:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{X}_{t+1}^{(i)} - X_{t+1}^{(i)}\right)^2,$$

where N represents the number of samples in the dataset, $\hat{X}_{t+1}^{(i)}$ denotes the predicted notional values of Bitcoin at time $t+1$ for the i-th sample, and $X_{t+1}^{(i)}$ are the corresponding true values. The first reason is that this is the most widely used and standard method in machine learning for this type of problem. The second reason is the metric we want to use to display the results, namely the R-squared R^2. The R^2 is interesting as a metric because it not only gives information on the error but also on the error given the variance of the estimated series, which means it's much easier to tell whether the model is performing well or not. It is also a measure widely used by econometricians and practitioners for this very reason. However, minimizing the MSE is exactly the same as maximizing the R^2, as its formula indicates:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(\hat{X}_{t+1}^{(i)} - X_{t+1}^{(i)})^2}{\sum_{i=1}^{N}(X_{t+1}^{(i)} - \bar{X}_{t+1})^2}.$$

As we can see, the upper terms of the quotient are equivalent to the sum of the squared errors, the other two components being fixed, the optimization of the mean squared error is totally similar to the optimization of R^2.

4.3.1 Note on training details

As in [GI24b] and [GI24a], metrics are calculated directly on scaled data and not on unscaled data. There are two reasons for this: firstly, MinMax scaling has no impact on the metric since the minimum is 0 and the data interval is $[0, 1]$; rescaling would simply have involved multiplying all the points, which would not have changed the R-squared. Regarding model optimization, we utilized the Adam optimizer, which is among the most popular options. Additionally, we designated 20% of our training set as a validation set and incorporated two training callbacks. The first is an early learning stop, which interrupts training after 6 consecutive periods without improvement on the validation set, and restores the weights associated with the best loss obtained on the validation set. The second is a reduction in the plateau learning rate, which halves the learning rate after 3 consecutive periods without improvement on the validation set. Finally, the sequence length given to all models for both tasks is set at a maximum between 45 and 5 times the number of steps forward. We set this lower limit at 45, as we observed that this positively improved the performance of all models. The results for predictions at 1, 3 and 6 steps ahead are therefore different from those in the TKAN and TKAT papers.

4.4 Task 1: Predicting Volumes

4.4.1 Model Architecture and Benchmarks

As these tasks are the same as those used in the TKAN and TKAT articles, we decided to use the same benchmarks for comparison. We also compared SigKAN with a variant called SigDense, in which the KANLinear is replaced by a fully connected dense layer, and the GRKAN is replaced by a standard GRN layer.

1. For the SigKAN and SigDense:
   (a) A variant with one SigKAN (resp. SigDense) layer with 100 units, which outputs is flatten and passed to a Dense 100 with activation 'relu' before a last linear layer. They are refered as SK-1 and SD-1 in the following tables.
   (b) A variant with two consecutive SigKAN (resp. SigDense) layers with 20 units, which outputs is flatten and passed to a Dense 100 with activation 'relu' before a last linear layer. They are refered as SK-2 and SD-2 in the following tables.

   For all of them, we used a signature level of 2, as this is the lowest level possible to capture the interaction between inputs, while also being the least expensive in terms of computation, which quickly becomes limiting due to the non-GPU support of signature operations.
2. For the TKAN, GRU and LSTM:
   (a) An initial recurrent layer of 100 units that returns complete sequences
   (b) An intermediate recurrent layer of 100 units, which returns only the last hidden state.
   (c) A final dense layer with linear activation, with as many units as there is timesteps to predict ahead

3. The TKAT use a number of hidden units of 100 in each layer, with the exception of the embedding layer, which has only one unit per feature. The number of heads used is 4.

4.4.2 Results

Results are separated into two tables. The first table 7.1 being the average R^2 obtained over five runs for SigKAN and SigDense. The second table 7.2 is the standard deviation of R^2 obtained over these five runs. To analyze the stability of the model in relation to its initialization over several runs.

Table 7.1 – R^2 Average: SigKAN and SigDense

Time	SK-1	SD-1	SK-2	SD-2
1	0.36225	0.33055	0.30807	0.31907
3	0.21961	0.21532	0.20580	0.21066
6	0.16361	0.15544	0.15351	0.15836
9	0.13997	0.12768	0.12684	0.13338
12	0.12693	0.11628	0.11826	0.11814
15	0.11861	0.11097	0.11448	0.11065

Table 7.2 – R^2 Average: Benchmarks

Time	TKAT	TKAN	GRU	LSTM
1	0.30519	0.33736	0.36513	0.35553
3	0.21801	0.21227	0.20067	0.06122
6	0.17955	0.13784	0.08250	-0.22583
9	0.16476	0.09803	0.08716	-0.29058
12	0.14908	0.10401	0.01786	-0.47322
15	0.14504	0.09512	0.03342	-0.40443

The average of R^2 shows that the proposed SigKAN model outperforms all the simple reference models, namely TKAN, GRU, and LSTM, while not reaching TKAT performance for this task over a longer period. The main advantage is that while TKAT and TKAN offer superior performance for one-step prediction, the SigKAN model outperforms all the other models. This indeed seems logical in the case of short-term predictions, given the very simple architecture of the SigKAN model compared to the Kolmogorov-Arnold time transformer. But it shows that, although it contains no recurrent calculus, it manages to outperform all the simple recurrent networks to which it is compared. The results also show that using the signature as a weighting mechanism is effective, since the SigDense and SigKAN versions outperform most recurrent models in the results. And, using KAN layers instead of dense layers shows an increase in performance, indicating that both components are interesting in the proposed layer. Finally,

we observe that stacking several layers does not seem to be effective in improving the performance of the KAN version on this task, while it improves that of the SigDense version on longer predictions. Using KAN in this case allows us to reduce the depth of our model for this case.

Table 7.3 – R^2 Standard Deviation: SigKAN and SigDense

Time	SK-1	SD-1	SK-2	SD-2
1	0.02378	0.01220	0.01102	0.01476
3	0.01435	0.00342	0.00319	0.00429
6	0.00376	0.00820	0.00438	0.00659
9	0.00346	0.01074	0.00665	0.00385
12	0.00369	0.00106	0.00830	0.00215
15	0.00171	0.00442	0.00216	0.00103

Table 7.4 – R^2 Standard Deviation: Benchmarks

Time	TKAT	TKAN	GRU	LSTM
1	0.01886	0.00704	0.00833	0.01116
3	0.00906	0.00446	0.00484	0.08020
6	0.00654	0.01249	0.02363	0.06271
9	0.00896	0.02430	0.01483	0.05272
12	0.00477	0.00132	0.08638	0.08574
15	0.01014	0.00701	0.02407	0.09272

What is particularly intriguing about the SigKAN model is its ability to produce more stable results across various runs, especially when compared to models based on recurrent networks for longer prediction tasks. So, while increasing performance compared to simple recurrent models, the addition of the signature component eliminates the need for the recurrent part, which is probably one of the drivers of the greatest instability in the other models.

Table 7.5 – R^2 Number of parameters: SigKAN and SigDense

Time	SK-1	SK-2	SD-1	SD-2
1	957,846	266,311	558,946	125,791
3	958,048	266,513	559,148	125,993
6	958,351	266,816	559,451	126,296
9	958,654	267,119	559,754	126,599
12	1,108,972	297,452	710,072	156,932
15	1,259,290	327,785	860,390	187,265

Table 7.6 – R^2 Number of parameters: Benchmarks

Time	TKAN	TKAT	GRU	MLP	LSTM
1	119,396	1,047,865	97,001	95,801	128,501
3	119,598	1,057,667	97,203	96,003	128,703
6	119,901	1,073,870	97,506	96,306	129,006
9	120,204	1,091,873	97,809	96,609	129,309
12	120,507	1,129,676	98,112	125,412	129,612
15	120,810	1,178,279	98,415	154,215	129,915

We believe it is valuable to consider the number of model parameters. Simpler than the TKAT models, the SigKAN model used here displayed a much higher number of parameters. This is due to the flattening of its output, which is transmitted to the last fully connected layer, rather than to the layer itself, which has a limited number of parameters. However, the KAN layer itself has its own number of parameters which can increase sharply with the number of inputs, so it is often wise to have a smaller input size to feed the networks. For example, with 15 prediction steps, the SigKAN layers contain 513,663 weights, while the remaining 751,615 are on the intermediate dense layers connected to the flattened output. Changing the architecture to avoid having such high-dimensional inputs could be a way of radically reducing the size of the model.

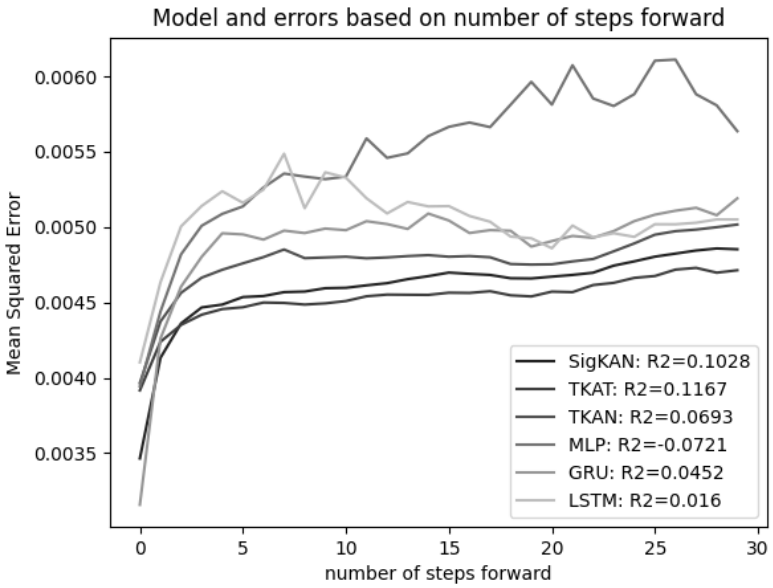


Figure 7.4 – Model and errors based on number of steps forward

Finally, we trained all models once to predict the next 30 steps, and displayed for each step the R^2 between the predictions and the actual values. As the results above show, SigKAN lies between TKAN and TKAT in terms of performance. There's also an MLP model included here, which is the one described in the following tasks, also tested for this comparison.

4.5 Predicting Absolute Returns: Benchmarks and Results

4.5.1 Model Architecture and Benchmarks

As the task here is different, with only one feature to predict, we have chosen to keep the same TKAN, GRU and LSTM benchmarks as before, but replace TKAT with a simple MLP model. The SigKAN architecture here refers to SigKAN with 100 hidden units as in the first task, and the MLP refers to a model where sequential inputs are first flattened, then passed through 2 dense layers of 100 units with ReLU activation, before being passed to a final linear unit with the number of steps forward to predict. Volatility being a highly autocorrelated process, we also added a very simple benchmark that uses the average of previous values as a predictor of the next. The window size selected for this benchmark is the best one between 1 and 50, chosen directly from the test set, which represents the best (unrealistic) result possible with this simple method.

4.5.2 Results

As before, the results are obtained over 5 runs for each and displayed in an average and standard deviation table.

Table 7.7 – R^2 Average: SigKAN versus Benchmarks

Time	SK-1	MLP	GRU	LSTM	TKAN	TKAT	Bench
1	0.1670	0.1458	0.1616	0.1581	0.09063	0.125312	0.1476
3	0.1497	0.1335	0.1452	0.1488	0.09063	0.132425	0.1387
6	0.1349	0.1232	0.1360	0.1246	-0.0351	0.132983	0.1293
9	0.1252	0.1172	0.1289	0.1288	0.00708	0.125162	0.1225
12	0.1164	0.1074	0.1163	0.1175	-0.0716	0.107760	0.1169
15	0.1165	0.1085	0.1229	0.1169	-0.0496	0.099868	0.1114

It appears that for this task, SigKAN is able to outperform all the other models on one-step-ahead predictions by a wide margin, while giving more sensible but no better predictions on longer-term predictions.The tasks also show that for these single-input tasks, the TKAN and TKAT models have much more difficulty than the other models, which may be a sign of less versatility than the SigKAN given the task. In contrast to LSTMs, which have results that are nearly the opposite of TKAN and TKAT, the latter two models achieve the weakest performance on the volume tasks but are able to considerably outperform the other models on the volatility task.

Table 7.8 – R^2 Standard Deviation: SigKAN versus Benchmarks

Time	SK-1	MLP	GRU	LSTM	TKAN	TKAT	Bench
1	0.0013	0.0050	0.0008	0.0003	0.0517	0.0558	0
3	0.0036	0.0080	0.0073	0.0012	0.0265	0.0087	0
6	0.0005	0.0037	0.0010	0.0020	0.0195	0.0022	0
9	0.0014	0.0026	0.0007	0.0022	0.0792	0.0028	0
12	0.0006	0.0074	0.0023	0.0006	0.0643	0.0060	0
15	0.0013	0.0057	0.0008	0.0001	0.0583	0.0070	0

What is also apparent is the real strength of this model: much greater stability, not only in terms of the tasks it can handle, but also in terms of stability over several calibrations, which is an important property. Finally, we also displayed the total number of parameters for each

Table 7.9 – R^2 Number of parameters: SigKAN versus Benchmarks

Time	SK-1	TKAN	TKAT	GRU	MLP	LSTM
1	563,646	113,906	485,743	91,601	14,801	121,301
3	563,848	114,108	495,545	91,803	15,003	121,503
6	564,151	114,411	511,748	92,106	15,306	121,806
9	564,454	114,714	529,751	92,409	15,609	122,109
12	714,772	115,017	567,554	92,712	17,412	122,412
15	865,090	115,320	616,157	93,015	19,215	122,715

model used, and we can see that, as before, the number of trainable parameters in SigKAN is much higher than in all the others, due to the flattening of the time dimension leading to a large, dense hidden layer. Here, for 15-step forward forecasting tasks, the number of weights held by the SigKAN layers is only 114,711, while the remaining 751,615 are due to the following layers. Here again, the reduction in the number of parameters could be achieved by modifying the way the layer output is used, by not just flattening it and fully connecting all outputs to the next hidden layer.

5 Conclusion

The adoption of path signatures, as an improvement on traditional KANs, was presented in this paper. It was achieved through an innovative combination with Kolmogorov-Arnold networks, a technique never before realized. This new development significantly improves the ability of these networks to handle tasks involving approximation functions thanks to the use of path signatures containing rich geometric information. The results of our experimental evaluations demonstrate that SigKANs not only outperform other conventional methods for multivariate time series data, but also offer a simple robust framework for modeling complex temporal relationships, which can be applied on a large scale without too much difficulty. The fusion itself holds great promise for future research and applications: it could find its place in a variety of

fields, including financial modeling or time series analysis, where such sophisticated tools are most needed. With this effort, we are going beyond mere academic interest in KANs; thanks to what we have developed here, new horizons can be opened up for machine learning, practical applications, and more generally, even other fields outside AI that could benefit from such developments.

Part IV

Time Series Need Attention

Chapter 8

An Attention Free Long Short-Term Memory For Time Series Forecasting

This part is a joint work with Ludovic De Villelongue (Univ. Paris Dauphine).

Abstract

Deep learning is playing an increasingly important role in time series analysis. We focus on time series forecasting using an Attention Free mechanism, a technique that has proven its efficiency. We propose a new architecture for forecasting nonlinear time series. We introduce a structure, based on Attention Free LSTM (AF-LSTM) layers, that surpasses linear models and some popular nonlinear models in order to forecast the conditional variances of cryptoassets. Our results validate the effectiveness of our model, which not only enhances the predictive ability of LSTM models, but also increases the efficiency of their learning process.

Contents

1 Introduction

As in most financial markets, the volatility in digital assets is largely influenced by the law of supply and demand, both being subject to real-world events. With digital assets, the effect of supply and demand is amplified due to a typical and regular lack of liquidity. Thus, the observed volatilities of digital assets returns are most often higher than traditional assets, at least for three reasons. First, the value of digital assets is highly dependent on their use. For example, for a project where a crypto-asset is used to obtain a service or reduce service usage fees, it will be more likely that its price will vary for some "structural" reasons, i.e. without being driven by agents who speculate solely on it. This is less the case for crypto whose use cases are more limited, as "meme coins". Second, digital assets are still relatively new compared to other assets. Various digital assets are being created day after day. These new digital assets can be traded on certain liquidity pools, but there is still very little activity. This lack of liquidity is driving volatility in the market. Third, as often with new asset classes, their prices are driven by opinion and "herding behaviors" (Abraham et al. [Abr+18] and Ante [Ant23]). The average digital assets investor is fairly inexperienced. This lack of experience can lead to irrational decisions that increase volatility. Some decentralized exchange platforms allow users to trade large amounts of coins without affecting market conditions; for example, "deep" liquidity pools, where funds are held for trading pairs, can help to absorb large orders without any significant price impact on the market. Decentralized exchanges thus represent only a small proportion of total market volumes, and then do not influence market volatilities in normal times. Moreover, the creation of new liquidity pools mechanisms leads to a shift towards a resilient and robust financial ecosystem. Nevertheless, as detailed in Inzirillo and De Quénetain [ID22a], some peculiar risks associated to such platforms may cause volatility spikes in the market.

Recently, we have seen an increase in interest towards developing attention based LSTM Wang et al. [Wan+16] and Zhang et al. [Zha+19]. Some researchers have focused on the development of particularly efficient mechanisms for prediction purposes, such as Attention Free architectures Zhai et al. [Zha+21a], an efficient variant of transformers Vaswani et al. [Vas+17]. The initial goal was to improve the performance of transformers. However, despite their complexity and flexibility, transformers do not seem to significantly improve prediction when dealing with time series Zeng et al. [Zen+22]. In this paper, we have retained the usefulness of attention mechanisms and more particularly their capacity to use the most relevant parts of the sequences of observations. We have combined this strength with the power of a LSTM (Hochreiter and Schmidhuber [HS97a]) to design a new architecture for time series prediction of highly volatile assets for which linear models are too crude to explain conditional volatilities. In our study, we focus on volatility modeling and forecasting by incorporating preliminary results from a GARCH model, enhancing its forecasts with an Attention Free Long-Short-Term Memory (AF-LSTM) approach. Our findings across different assets demonstrate that using the AF-LSTM layer improves the efficiency of time series prediction.

2 Systematic Volatility

Naively, a standard assumption that is often made in linear models is homoscedasticity: residuals are centered and their standard deviation given past values is a constant σ. The latter assumption is not realistic in the case of financial applications. In practice, their innovations are heteroscedastic, even if the strength of this feature depends on every asset class. In other words, conditional asset return variances are generally time dependent. This can be observed historically with the well-known presence of volatility clusters. Volatility clustering is typically explained by conditional heteroscedasticity Bollerslev [Bol86].

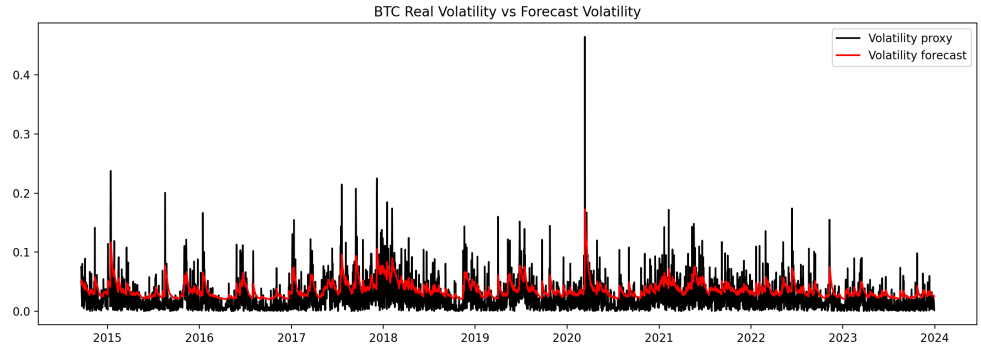


Figure 8.1 – Bitcoin volatility vs forecasted volatility using GARCH(1,1)

This phenomenon is particularly strong in the case of digital assets. As a first historical attempt of building heteroskedastic models for asset returns, in ARCH models, conditional variances are linear functions of a fixed number of squared past returns. Despite its appealing features, such as simplicity, nonlinearity, easiness of inference and adjustment for forecasting, ARCH models suffer from some drawbacks: equal responses to positive and/or negative shocks, relatively strong model assumptions and relatively poor predictive performances. But, last but not least, conditional volatilities of ARCH models depend on a fixed number of past returns and not on the whole observed path. To overcome such limitations, ARCH models have been extended with GARCH models (Bollerslev [Bol86]). As ARCH models, the latter family is also an attempt to capture the volatility dynamics of financial assets that exhibit now autoregressive features. As a consequence, GARCH-type conditional volatilities depend on the whole path of past returns. Due to their nice empirical performances, GARCH models have been seen for decades as one of the leading tools in financial econometrics, especially in the case of financial returns with time dependent volatilities.

Formally, a series $(r_t)_{t\in\mathbb{Z}}$ of asset returns follows GARCH(p,q) dynamics when there is a sequence of innovations $(\epsilon_t)_{t\in\mathbb{Z}}$ that are martingale differences and such that, for every t,

$$r_t = \sigma_t \epsilon_t, \quad E[\epsilon_t | r_{t-1}, r_{t-2}, \ldots] = 0, \; E[\epsilon_t^2 | r_{t-1}, r_{t-2}, \ldots] = 1,$$

$$\sigma_t^2 = \omega + \sum_{j=1}^{p} \alpha_j r_{t-j}^2 + \sum_{k=1}^{q} \beta_k \sigma_{t-k}^2, \tag{8.1}$$

where $\sum_{j=1}^{p} \alpha_j + \sum_{k=1}^{q} \beta_k < 1$. The law of the innovations is often assumed to be centered Gaussian. As a consequence of 8.1, the time-varying variance process (σ_t^2) fluctuates around the unconditional variance that is given by

$$\sigma^2 := Var(r_t) = \frac{\omega}{1 - \sum_{j=1}^{p} \alpha_j - \sum_{k=1}^{q} \beta_k}.$$

The Glosten-Jagannathan-Runkle GARCH Glosten, Jagannathan, and Runkle [GJR93], denoted GJR-GARCH, is a variant of the latter standard GARCH model. With GJR-GARCH specifications, negative shocks in a previous period have typically a stronger impact on the current conditional variance than positive shocks. Therefore, GJR-GARCH volatility dynamics are often defined as follow:

$$\sigma_t^2 = \omega + \sum_{j=1}^{p} \alpha_j r_{t-j}^2 + \sum_{j=1}^{p} \gamma_j r_{t-j}^2 \mathbf{1}_{\{r_{t-j}<0\}} + \sum_{k=1}^{q} \beta_k \sigma_{t-k}^2, \tag{8.2}$$

where $r_t = \sigma_t \epsilon_t$ as for GARCH specifications. Many other extensions and refinements of standard GARCH models have been proposed in the literature: Exponential GARCH (EGARCH), Threshold GARCH (TGARCH), Asymmetric Power GARCH (APARCH), etc. See Francq and Zakoian [FZ19] (Chapter 10) for an up-to-date presentation.

3 LSTM Needs Focus

With the expanding need to improve the accuracy of volatility modeling, GARCH models -despite their nice properties- have been challenged in the financial and economic literature. Indeed, the relationship between conditional variances and squared returns being probably non linear, it is tempting to introduce some non linearity in these models. This was the approach of many researchers (Nikolaev, Tino, and Smirnov [NTS13] and Kristjanpoller and Minutolo [KM15], among others) who tried to introduce non-linearities into "classical" models by relying on neural networks. In particular, [KW18] or [LS20] mixed a LSTM with multiple GARCH-type models to introduce non linear features. In this section, we seek to improve the predictive ability of GARCH models by relying on Attention Free mechanims Zhai et al. [Zha+21a]. We do not question the estimation of the model (as in most applied papers in computer science). Our efforts will rather be to improve the instantaneous volatility predictions that will be made at each time step. To this aim, we recover the conditional variances obtained after a first stage estimation through a univariate GARCH model. We propose to improve the predictive capacity of GARCH-type models by adding an extra step using deep learning models. This should allow us to estimate the conditional volatility of cryptocurrencies even more accurately. Our idea is based on the following observation: for highly volatile time series, GARCH models can sometimes face challenges to identify and modelize some specific volatility patterns, due to their relatively constrained structure and assumptions. Our idea will be to use the conditional volatilities obtained by a GARCH model as an input to a Recurrent Neural Network (RNN). Concerning the latter network, we propose an extension of an Long Short Term Memory (LSTM).

LSTM Hochreiter and Schmidhuber [HS97a] are RNNs with gated mechanisms that are designed to avoid vanishing gradients. The blocks of LSTM contain three non-linear gates (contrary to classical neuron-based neural networks), as well as a "memory" for the management of some sequences of intermediate features. The three types of non-linear gates are denoted i_t, f_t and o_t, respectively. The input gate i_t plays an important role in filtering significant values to update the memory states. The forget gate f_t determines which part of information has to be deleted at every time t, setting its value to 0. This allows the model to regularly forget irrelevant pieces of information for forecasting, due to regular updates. The output gate o_t is used to obtain the output of the LSTM.

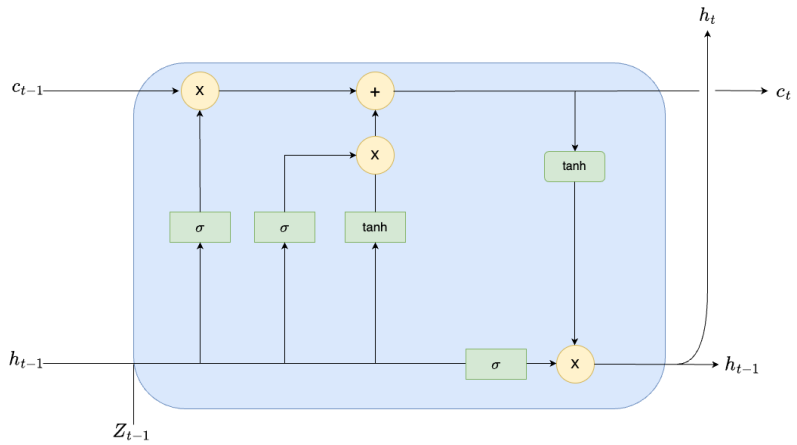


Figure 8.2 – LSTM Cell

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, Z_{t-1}] + b_f),\\
i_t &= \sigma(W_i[h_{t-1}, Z_{t-1}] + b_i),\\
\tilde{c}_t &= \tanh(W_c[h_{t-1}, Z_{t-1}] + b_c),\\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t,\\
o_t &= \sigma(W_o[h_{t-1}, Z_{t-1}] + b_o),\\
h_t &= o_t \odot \tanh(c_t).
\end{aligned}
\tag{8.3}
$$

The set of parameters to be calibrated is $\theta := \{W_f, W_i, W_c, W_o, b_f, b_i, b_c, b_o\}$, and Z_{t-1} is the input vector at time t. At each time step, the memory cell takes the current information Z_{t-1}, the previous hidden state h_{t-1} and the previous memory state c_{t-1}. The two latter quantities will be updated. The ultimate goal is to predict some quantity y_t based on past returns, and possibly some additional relevant information (known before t). Typically, y_t could be as the asset return at time t itself (a difficult task), or the direction of the market (a binary variable indicating whether the t-return is positive or negative), or the instantaneous volatility at t, etc. Our predictor will be denoted $\hat{y}_t$. In general, it is a complex non linear function of $z_{t-1}, z_{t-2}, \ldots$ and the model parameter θ. In this paper, we try to predict squared returns, i.e.

$$y_t := \mathbb{E}[r_t^2 | \mathcal{F}_{t-1}],$$

denoting $\mathcal{F}_{t-1}$ our set of past information. In the latter sigma-algebra, we will choose some past conditionnal volatilities estimated using a GARCH(1,1) as well some past returns. In other words, we impose

$$y_t := \mathbb{E}[r_t^2 | \sigma_{t-1}^2, ..., \sigma_{t-p}^2, r_{t-1}, ..., r_{t-p}], \tag{8.4}$$

for some constant p. The sequence lenght of our inputs will be fixed by a parameter denoted q, the maximum lag. In the latter section, our input sequences, a subset of $\mathcal{F}_{t-1}$ will be denoted Z_{t-1}. $Z_{t-1} \in \mathbb{R}^{p \times q}$ is a stack vector of past information such as:

$$Z_{t-1} := \{\Sigma_{t-1}, R_{t-1}\},$$

where $\Sigma_{t-1} \in \mathbb{R}^p$ and $R_{t-1} \in \mathbb{R}^p$. Before introducing our innovating RNN, an overview of the so-called attention mechanisms cannot be avoided. Google introduced the transformers (Vaswani et al. [Vas+17]), an attention based model which has made it possible to capture long range patterns/memory. The main distinction between attention mechanisms and traditional RNN or LSTM models lies in the way they process long sequences of data. Traditional recurrent models such as RNNs process the sequence sequentially, processing elements one after the other. Attention mechanisms, on the other hand, allow the model to focus directly on and prioritise specific, relevant parts of the long sequence, rather than necessarily going through the whole sequence. This mechanism makes possible to "capture information at very early positions in the sequence" Lezmi and Xu [LX23]. The Attention Free Transformer (Zhai et al. [Zha+21a]) is an extension of transformers, proposed by a group of researchers from Apple. They introduce "attention free blocks", an alternative to the "attention blocks" which eliminates the need for dot product self attention, as detailed in (8.5)-(8.8).

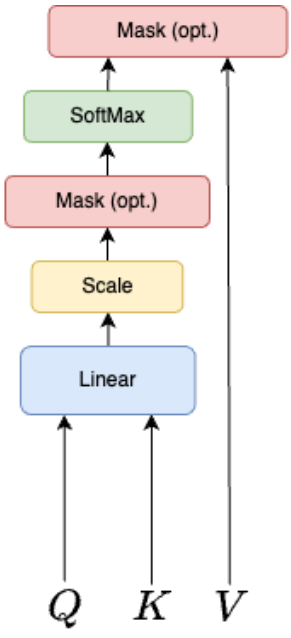


Figure 8.3 – Scale Dot-Product Attention

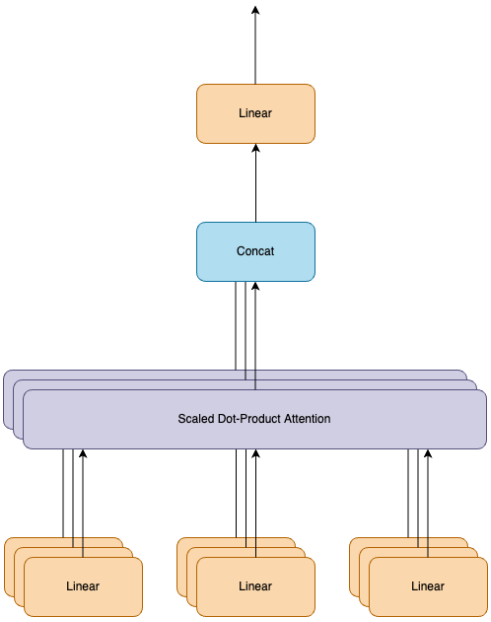


Figure 8.4 – Multi-Head Attention

Scaled Dot-Product Attention used in Figure 8.3-8.4 is similar to Dot-Product Attention introduced by Luong, Pham, and Manning [LPM15] and self dot product attention used in Transformers. We denote $Q \in \mathbb{R}^{m \times d_k}$, $K \in \mathbb{R}^{n \times d_k}$, and $D \in \mathbb{R}^{n \times d_v}$ the so-called "queries", "keys" and "values" respectively. Scaled Dot-Product Attention is

$$\text{Attention}(Q, K, V) := \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right).$$

In terms of computation steps, start with Q and K, our Query and Key matrix, respectively:

$$Q := \begin{pmatrix} q_{0,0} & q_{0,1} & \cdots & q_{0,d_k} \\ q_{1,0} & q_{1,1} & \cdots & q_{1,d_k} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m,0} & q_{m,1} & \cdots & q_{m,d_k} \end{pmatrix} = \begin{pmatrix} \vec{q}_0 \\ \vec{q}_1 \\ \vdots \\ \vec{q}_m \end{pmatrix}, \tag{8.5}$$

$$K^T := \begin{pmatrix} \vec{k}_0 & \vec{k}_1 & \cdots & \vec{k}_n \end{pmatrix}. \tag{8.6}$$

To obtain our attention weights, we divide each element of QK^T (8.7) by $\sqrt{d_k}$ and apply the softmax function per row:

$$QK^T = \begin{pmatrix} \vec{q}_0 \cdot \vec{k}_0 & \vec{q}_0 \cdot \vec{k}_1 & \cdots & \vec{q}_0 \cdot \vec{k}_n \\ \vec{q}_1 \cdot \vec{k}_0 & \vec{q}_1 \cdot \vec{k}_1 & \cdots & \vec{q}_1 \cdot \vec{k}_n \\ \vdots & \vdots & \ddots & \vdots \\ \vec{q}_m \cdot \vec{k}_0 & \vec{q}_m \cdot \vec{k}_1 & \cdots & \vec{q}_m \cdot \vec{k}_n \end{pmatrix}, \tag{8.7}$$

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \begin{pmatrix} \text{softmax}\left(\frac{\vec{q}_0 \cdot \vec{k}_0}{\sqrt{d_k}}, \frac{\vec{q}_0 \cdot \vec{k}_1}{\sqrt{d_k}}, \ldots, \frac{\vec{q}_0 \cdot \vec{k}_n}{\sqrt{d_k}}\right) \\ \text{softmax}\left(\frac{\vec{q}_1 \cdot \vec{k}_0}{\sqrt{d_k}}, \frac{\vec{q}_1 \cdot \vec{k}_1}{\sqrt{d_k}}, \ldots, \frac{\vec{q}_1 \cdot k_n}{\sqrt{d_k}}\right) \\ \vdots \\ \text{softmax}\left(\frac{\vec{q}_m \cdot \vec{k}_0}{\sqrt{d_k}}, \frac{\vec{q}_m \cdot \vec{k}_1}{\sqrt{d_k}}, \ldots, \frac{\vec{q}_m \cdot \vec{k}_n}{\sqrt{d_k}}\right) \end{pmatrix} = \begin{pmatrix} s_{0,0} & s_{0,1} & \cdots & s_{0,n} \\ s_{1,0} & s_{1,1} & \cdots & s_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m,0} & s_{m,1} & \cdots & s_{m,n} \end{pmatrix}, \tag{8.8}$$

where, for each row i, as a result of the softmax operation, we have $\sum_{j=0}^{n} s_{i,j} = 1$. The last step is to multiply this matrix by V.

Transformers are known as efficient tools to capture long-term time dependencies by focusing on some relevant parts of the studied sequence. Just like transformers, the attention-free transformers (AFT) use attention, including interactions between queries, keys, and values. The main difference between both families comes from the fact that an Attention Free (AF) block firstly combines key and value together with a set of learned position biases, as described in Zhai et al. [Zha+21a]. Then the query is combined with the context vector. As explained in [Zha+21a]: *"Explained in simple words, for each target position t, AFT performs a weighted average of values, the result of which is combined with the query with element-wise multiplication. In particular, the weighting is simply composed of the keys and a set of learned pairwise position biases. This provides the immediate advantage of not needing to compute and store the*

expensive attention matrix while maintaining the global interactions between query and values as Multi-Head Attention does."

In this paper the idea is to combine the use of "attention free blocks" with the power of LSTMs for time series forecasting, that outform traditional econometrics models Siami-Namini, Tavakoli, and Namin [STN18]. We added an "attention free block" to build an Attention Free LSTM layer (AF-LSTM). The AF-LSTM layer processing step can be split in several equations. The input at each time step will firstly be processed by an Attention Free layer defined as

$$\tilde{Z}_{t-1}^{(l)} = AF_1^{(l)}(W_1^{(l)}; Z_{t-1}^{(l)}), \tag{8.9}$$

where $Z_{t-1}^{(l)} \in \mathbb{R}^p$ is the input sequence of the l-th layer, and $AF_1^{(l)}$ denotes the Attention Free encoding map associated to the l-th layer. This map is detailed as follows:

$$\begin{aligned}
x_{t-1}^{(l)} &= \text{Concat}(W_x^{(l)} Z_{t-1}^{(l)} + b_x^{(l)}), \\
Q_{t-1}^{(l)} &= W_q^{(l)} x_{t-1}^{(l)}, \\
K_{t-1}^{(l)} &= W_k^{(l)} x_{t-1}^{(l)}, \\
V_{t-1}^{(l)} &= W_v^{(l)} x_{t-1}^{(l)}, \\
\eta_{t-1}^{(l)} &= \sum_{t'=1}^{p} \text{Softmax}(K_{t't-1}^{(l)}) \odot V_{t',t-1}^{(l)}, \\
\tilde{Z}_{t-1}^{(l)} &= \sigma_q(Q_{t-1}^{(l)}) \odot \eta_{t-1}^{(l)}.
\end{aligned} \tag{8.10}$$

where $\sigma_q(.)$ is the nonlinearity applied to the query with default being sigmoid, p is the sequence lenght of the input corresponding to the lag values of GARCH model specification. We then apply the *Layer Normalization* LayerNorm(.) function Ba, Kiros, and Hinton [BKH16] to the transformed input $\tilde{Z}_{t-1}^{(l)}$, i.e. the transform

$$\text{LayerNorm}(x; \psi; \phi) = \psi \frac{(x - \mu_x)}{\sigma_x} + \phi,$$

where (ψ, ϕ) is a set of learnable parameters. After the layer normalization stage, we apply a filter using the $ReLU$ activation function, $ReLU(x) = max(0, x)$. Hence, we obtain

$$\tilde{z}_{t-1}^{(l)} = \text{ReLU}\big(\text{LayerNorm}(\tilde{Z}_{t-1}^{(l)}; \psi_1^{(l)}; \phi_1^{(l)})\big). \tag{8.11}$$

On the right hand side of Figure 8.5, the inputs are the same as on the left hand side. The ouput of the two channels are denoted $\tilde{z}_{t-1}^{(l_f)}$ and $\bar{z}_{t-1}^{(l)}$ respectively. We denote by $\bar{Z}_{t-1}$ the output of a second Attention Free mechanism $AF_2^{(l)}$, which resume the information of the input tensor. Hence, this quantity be multiplied by the ouput of the left side to only select the observations that will have a significant impact for prediction purpose:

$$\begin{aligned}
\bar{Z}_{t-1}^{(l)} &= AF_2^{(l)}(W_2^{(l)}; Z_{t-1}^{(l)}), \\
\bar{z}_{t-1}^{(l)} &= \text{ReLU}\big(\text{LayerNorm}(\bar{Z}_{t-1}; \psi_2^{(l)}; \phi_2^{(l)})\big).
\end{aligned} \tag{8.12}$$

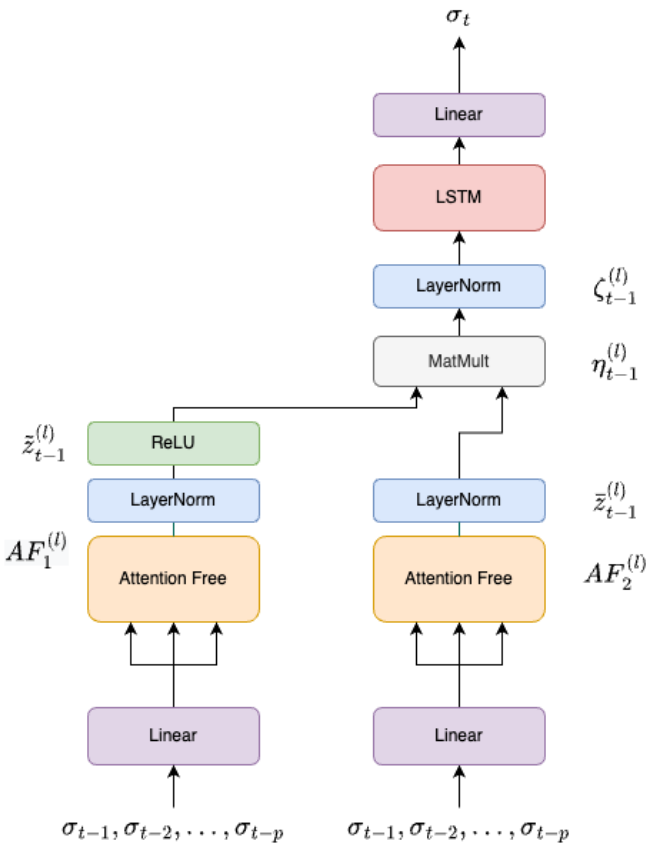


Figure 8.5 – Attention Free LSTM For Volatility Forecasting

The output of the two channels will then be multiplied, which allow us to filter the input information:

$$\begin{aligned} \eta_{t-1}^{(l)} &= \tilde{z}_{t-1}^{(l)} \odot \bar{z}_{t-1}^{(l)}, \\ \zeta_{t-1}^{(l)} &= \text{LayerNorm}(\eta_{t-1}^{(l)}; \psi^{(l)}; \phi^{(l)}). \end{aligned} \tag{8.13}$$

$\zeta_{t-1}^{(l)}$ will be the input of a parametric function $g_w^{(l)}$ to yield updated hidden states:

$$h_t^{(l)} = g_w^{(l)}(\zeta_{t-1}^{(l)}). \tag{8.14}$$

The AF layer is described in Figure 8.5. The global LSTM architecture yielding $g_w^{(l)}(.)$ is detailed in Figure 8.2. Using the output $g_w^{(l)}(.)$, we can estimate the output of the final layer (whose index is l_f) to obtain $\sigma_t^{(l_f)}$ given by

$$\sigma_t^{(l_f)} = W_y^{(l_f)} h_{t-1}^{(l_f)} + b_y^{(l_f)}. \tag{8.15}$$

4 Experiment

The retrieved data are the Bitcoin daily closing prices of these digital assets since its historical quotes are available. We also made the experiment on other cryptocurrencies (ETH, SOL,LTC, etc.), results are available in the appendix 6. The series of log returns related to Bitcoin prices is calculated and fed to the GARCH(1,1) model

$$\begin{aligned} r_t &= \sigma_t \epsilon_t, \\ \sigma_t^2 &= \omega + \alpha r_{t-1}^2 + \beta \sigma_{t-1}^2, \end{aligned} \tag{8.16}$$

The distribution of the standardized residuals is assumed to be standard Gaussian. Once the latter GARCH model is fitted by Quasi Maximum Likelihood, the estimated conditional variances (σ_t^2) are saved in a vector Σ_t. The lagged values of this vectors will be stacked with the vector of past returns within $Z_{t-1} := \{\Sigma_{t-1}, R_{t-1}\}$. Our dataset of inputs Z_{t-1} will be splited into two sub-datasets (train subset and test subset).

Concerning the LSTM network, we choose the following parameters:

— *Input size*: 2 (number of expected input features),

— *Hidden size*: 64 (number of features in hidden states h).

The AF-LSTM network is built with the following parameters,

— *Input size*: 2, (number of expected features as inputs),

— *Hidden size*: 64 (number of features in hidden states h),

— *Maximum sequence length*: is the is the same size of the sequence length. (maximum number of time steps),

— *Dim*: 2 (embedding dimension of the keys, queries and values).

After adjusting the LSTM and AF-LSTM networks using an Adam optimizer, we calculate their mean squared error losses over the train set. In the second time, we used the test sest (out of sample) to assess the performance of the model. We expect the AF-LSTM should be able to catch some jumps due to its ability to focus on any part of long sequences of inputs, in terms of metrics, we would expect a lower residual mean squared error using the AF-LSTM. Our estimation and forecast have been made on top 5 digital assets in term of market capitalization, in this sub section we will discuss only Bitcoin but ETH, SOL, LTC and BNB table of results are available in the appendix 6.

Bitcoin

Table 8.1-8.2 shows the forecasting performance of the LSTM and AF-LSTM, for predicting Bitcoin squared returns values. The LSTM model shows consistent performance across the three time steps. LSTM shows low Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) values. In contrast, the AF-LSTM model appears to have even lower MSE and RMSE. This indicates superior forecasting accuracy compared to the LSTM model. However,

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0058	0.0763	0.0652	11.2880
t+2	0.0055	0.0744	0.0634	10.9141
t+3	0.0056	0.0746	0.0637	11.0528

Table 8.1 – Forecast using LSTM (Bitcoin out of sample).

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0012	0.0349	0.0253	16.0520
t+2	0.0012	0.0352	0.0255	16.8245
t+3	0.0012	0.0352	0.0256	17.1902

Table 8.2 – Forecast using AF-LSTM (Bitcoin out of sample).

the Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) are higher for the AF-LSTM model. Overall, both models demonstrate promising results in predicting the future squared values of Bitcoin, with the AF-LSTM showing potential advantages in terms of minimizing the overall error metrics.

5 Conclusion

A GARCH(1,1) model updates instantaneous volatilities based on the most recent observed returns and smoothly (due to the autoregressive feature of conditional volatility dynamics). This may be satisfying if price changes are of "reasonable" size. However, this model's weakness emerges when there are more sudden jumps in terms of log returns. To consider the impact of these shocks on volatility levels, an LSTM model could be used, as it tries to mimic the real data features by working with many lagged values. To boost its performance and increase its prediction power, using Attention Free LSTM layers can be very useful. It allows catching the jumps in volatility in a better way than LSTM. The latter approach eliminates the quadratic complexity of the self-attention mechanism while selecting the past observations that will be relevant for prediction purposes. In fact, instead of carrying out the dot product for the creation of the attention matrix, a weighted average of the values within sequences is carried out for each target position. Using the "Attention Free" mechanism maintains all the advantages of the dot product without the computational cost.

6 Appendix

6.1 Ethereum

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0058	0.0765	0.0654	11.3247
t+2	0.0055	0.0744	0.0633	10.8294
t+3	0.0058	0.0759	0.0651	11.2975

Table 8.3 – Forecast using LSTM (Ethereum out of sample).

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0011	0.0335	0.0287	6.0659
t+2	0.0011	0.0331	0.0280	5.9049
t+3	0.0012	0.0342	0.0294	6.2497

Table 8.4 – Forecast using AF-LSTM (Ethereum out of sample).

6.2 Forecasting on Solana

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0058	0.0763	0.0651	11.2140
t+2	0.0056	0.0749	0.0638	10.9530
t+3	0.0059	0.0767	0.0659	11.4337

Table 8.5 – Forecast using LSTM (Solana out of sample).

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0049	0.0698	0.0523	8.1054
t+2	0.0049	0.0701	0.0524	8.2302
t+3	0.0049	0.0699	0.0515	7.8602

Table 8.6 – Forecast using AF-LSTM (Solana out of sample).

6.3 Forecasting on Litecoin

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0060	0.0772	0.0660	11.3860
t+2	0.0057	0.0752	0.0642	11.0121
t+3	0.0058	0.0759	0.0650	11.2361

Table 8.7 – Forecast using LSTM (Litecoin out of sample).

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0024	0.0487	0.0333	6.4853
t+2	0.0024	0.0487	0.0337	6.5818
t+3	0.0024	0.0490	0.0343	6.6590

Table 8.8 – Forecast using AF-LSTM (Litecoin out of sample).

6.4 Forecasting on BNB

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0058	0.0759	0.0646	11.1581
t+2	0.0056	0.0748	0.0637	10.9136
t+3	0.0057	0.0754	0.0645	11.1930

Table 8.9 – Forecast using LSTM (BNB out of sample).

Horizon	MSE	RMSE	MAE	MAPE
t+1	0.0010	0.0311	0.0242	7.6578
t+2	0.0010	0.0320	0.0254	8.2723
t+3	0.0011	0.0329	0.0268	8.6988

Table 8.10 – Forecast using AF-LSTM (BNB out of sample).

Chapter 9

An Attention Free Conditional Autoencoder For Anomaly Detection in Cryptocurrencies

This part is a joint work with Ludovic De Villelongue (Univ. Paris Dauphine).

Abstract

Detecting anomalies in time series is challenging due to the presence of significant noise. While denoising techniques can reduce this noise, they often result in substantial information loss. To address this, we introduce an Attention Free Conditional Autoencoder (AF-CAE) for anomaly detection in time series data. Our approach builds on a simple conditional autoencoder model by incorporating an Attention Free LSTM layer, enhancing the reliability and effectiveness of anomaly detection Inzirillo and De Villelongue [ID22b]. Comparative results demonstrate that our AF-CAE outperforms the LSTM conditional autoencoder, offering improved model explanatory power and superior anomaly detection in noisy time series.

Contents

1 Introduction

To identify anomalies in prices on a basket of cryptocurrencies returns, autoencoders can be useful [Pol+19; ZP17; SY14]. An autoencoder is a type of neural network in which outputs try to mimic input variables. It is divided into two steps: in the first one (the "encoding part"), input variables are passed inside a certain number of neurons in the hidden layers, creating a compressed representation of them; in the second step (the "decoding part") unpacks the latter input representation and maps it to an output layer that tries to recover the inputs. An autoencoder with no other variable than inputs can thus be qualified as a dimension reduction and unsupervised learning device. Self-attention mechanisms proposed by Zhai et al. [Zha+21a] embedded within a Transformer allow us to increase the efficiency of the learning tasks in machine translation, language understanding, and other paradigms. We introduce an Attention Free LSTM Conditional Autoencoder (AF-CAE), that is able to focus on input sequences that will have an impact on predictions. In a later paper Gu, Kelly, and Xiu [GKX21] introduced nonlinear functions to identify latent factors of random variables. However, they use LSTM layers, which turn out to be less effective than models based on attention mechanisms [Zey+19]. In this paper, we look for an architecture to detect anomalies which led us to identify the importance of factors of our time series. Since Zhai et al. [Zha+21a] found an Attention Free Transformer (AFT) more efficient than usual transformers [Vas+17] during the learning task, we started from this point. We then added an attention-free transformer in the encoder-decoder structure to only capture the variables that have a positive impact on the prediction and minimize the reconstruction error. The Attention Free Autoencoder structure in a later section better identifies anomalies in time series.

2 Factor Models

2.1 Linear Factor Models

When an autoencoder has a single hidden layer and a linear activation function, it is equivalent to a PCA estimator for linear factor models. The static linear factor model is written

$$y_t = \beta f_t + u_t, \tag{9.1}$$

where $(Y_t)_{0\leq t\leq T}$ is a vector of random variables, y_t is a real random variable and f_t is a $K \times 1$ vector of factors, u_t is a $N \times 1$ vector of idiosyncratic errors (independent of f_t), and β is the $N \times K$ matrix of factor loadings modelled as a nonlinear function of covariates. The matrix form of the factor model is given by

$$Y = \beta F + U. \tag{9.2}$$

2.2 Non Linear Factor Models

Gu, Kelly, and Xiu [GKX21] proposed a new autoencoder-based approach to estimate latent factors for financial time series. Inspired by a work of Kozak [Koz19], they introduced nonlinear

functions to estimate nonlinear conditional exposures and latent factors associated. A gain in precision can be obtained by incorporating asset-specific covariates in the specification of factor loadings (Gu, Kelly, and Xiu [GKX21]). These covariates also indirectly improve the estimates of the latent factors themselves. Their model formulation amounts to a combination of two linear models: Latent factors are specified linearly and the factor loadings are modeled as dependent on covariates, is described by the equation:

$$y_{i,t} = \beta(X_{i,t-1})f_t + u_{i,t}. \tag{9.3}$$

Additionally, a linear approach is adopted for conditional betas, representing factor loadings as portfolios of individual stock returns, formulated as:

$$\beta(X_{i,t-1}) = X_{i,t-1}\Gamma. \tag{9.4}$$

where $X_{i,t-1}$ denotes the asset characteristics.

3 Autoencoders

Autoencoders Hinton and Salakhutdinov [HS06] are a family of neural networks trained to reconstruct their input, which forces them to learn a compressed representation of the data. "Autoencoding" is then an unsupervised learning task that can be used for dimensionality reduction as well as feature extraction. There is a close connection between autoencoders and PCA [BH89] and, by extension, between autoencoders and latent factor asset pricing models Gu, Kelly, and Xiu [GKX21].

3.1 Simple Linear Autoencoder

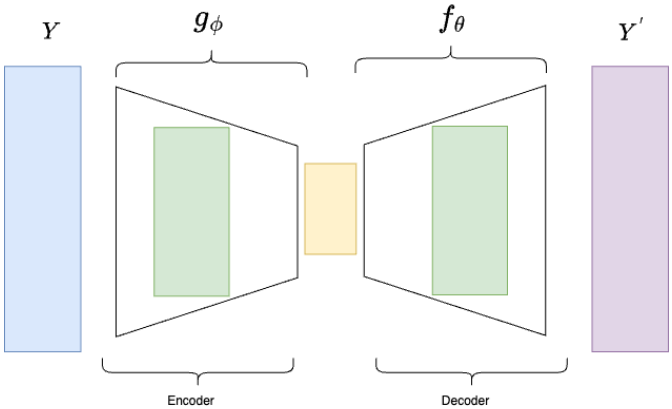


Figure 9.1 – Linear Autoencoder

Let us define ϕ and θ the encoder and decoder set of parameters, respectively. Where $\phi := \{\phi^{(0)}, \phi^{(1)}\}$ and $\theta := \{\theta^{(0)}, \theta^{(1)}\}$ We can write the one-layer, linear autoencoder with K

neurons as

$$\begin{aligned} \hat{y}_t &= f_\theta(g_\phi(y_t)) + u_t, \\ &= \theta^{(0)} + \theta^{(1)}(\phi^{(0)} + \phi^{(1)} y_t) + u_t, \end{aligned} \tag{9.5}$$

where $y_t = log(P_t) - log(P_{t-1})$ a N dimensional vector of log returns, $\phi^{(1)}$, $\theta^{(1)}$, $\theta^{(0)}$ and $\phi^{(0)}$ are $K \times N$, $N \times K$, $N \times 1$, and $K \times 1$ matrices of parameters, respectively. The objective of the model is to minimize the reconstruction error u_t for each time step, defined as

$$u_t = y_t - f_\theta \left(g_\phi(y_t) \right). \tag{9.6}$$

The parameters of the encoder and the decoder ϕ and θ can be estimated by solving the following optimization problem:

$$\min_{\phi,\theta} \sum_{t=1}^{T} \left\| y_t - \left(\theta^{(0)} + \theta^{(1)}(\phi^{(0)} + \phi^{(1)} y_t) \right) \right\|^2. \tag{9.7}$$

3.2 Conditional Autoencoder Model

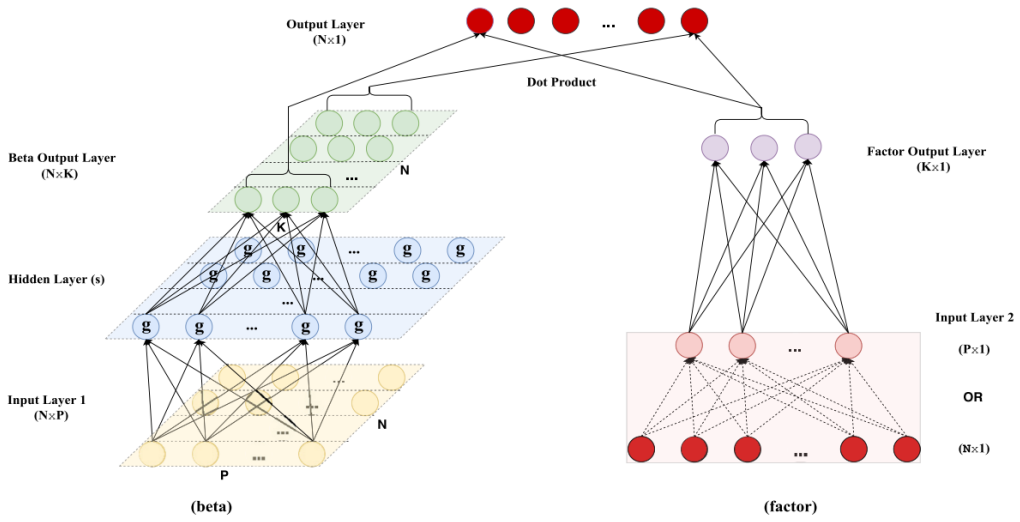


Figure 9.2 – Conditional Autoencoder Model Gu, Kelly, and Xiu [GKX21]

Figure 9.2 illustrates the structure of the conditional autoencoder introduced by Gu, Kelly, and Xiu [GKX21]. The left side of the network builds factor loadings as a nonlinear function of covariates. These covariates are typically some characteristics of the time series, such as technical indicators. On the right-hand side, the network specifies factors as portfolios of individual cryptocurrency returns. The model can then be written as

$$y_{i,t} = \beta_{i,t-1} f_t + u_{i,t}, \tag{9.8}$$

where the recursive formulation for the nonlinear beta function is

$$\begin{aligned} X_{i,t-1}^{(l)} &= g(b^{(l-1)} + W^{(l-1)} X_{i,t-1}^{(l-1)}),\ l = 1, ..., l_\beta, \\ \beta_{i,t-1} &= b^{(L_\beta)} + W^{(L_\beta)} X_{i,t-1}^{(l_\beta)}. \end{aligned} \tag{9.9}$$

Moreover, the recursive mathematical formulation of the factors is

$$\begin{aligned} y_t^{(0)} &= y_t, \\ y_t^{(l)} &= \tilde{g}(\tilde{b}^{(l-1)} + \tilde{W}^{(l-1)} y_t^{(l-1)}),\ l = 1, ..., l_f, \\ X_t &= \tilde{b}^{(l_f)} + \tilde{W}^{(l_f)} y_t^{(l_f)}. \end{aligned} \tag{9.10}$$

3.3 Attention Free Conditional Autoencoders (AF-CAEE)

The Conditional Autoencoder proposed by Gu, Kelly, and Xiu [GKX21]

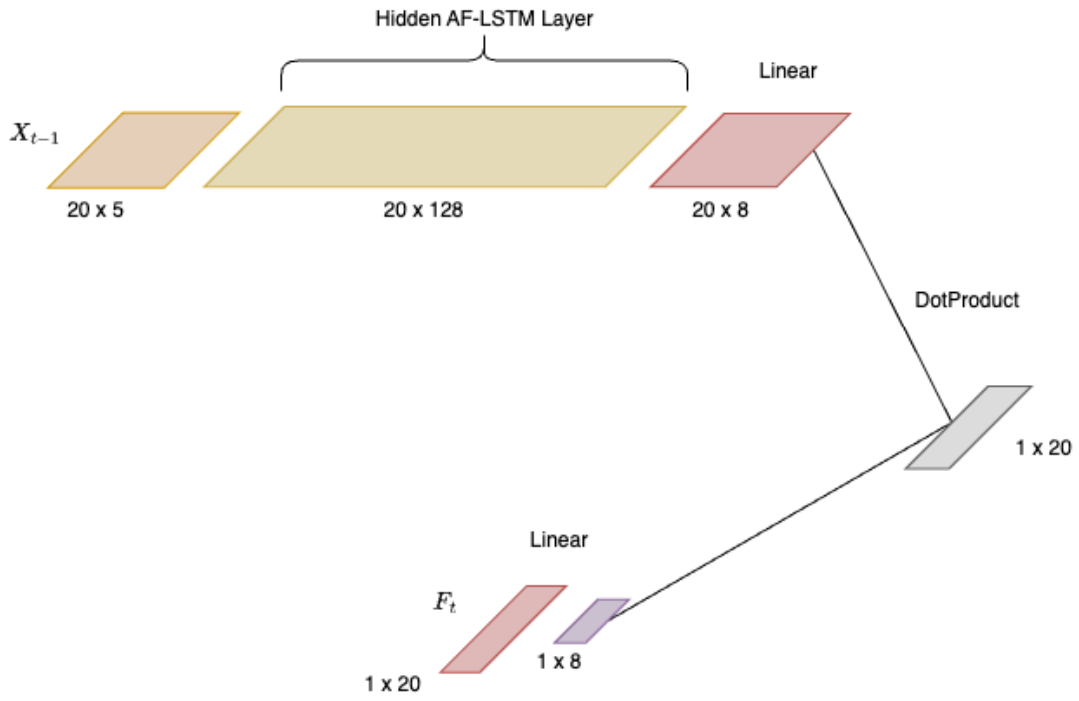


Figure 9.3 – Conditional Attention Free Autoencoder

LSTM

The LSTM framework Hochreiter and Schmidhuber [HS97a] is RNNs with gated mechanisms designed to avoid vanishing gradient. The blocks of LSTM contain 3 nonlinear gates; *input gate*, *forget gate* and *output gate*. This architecture embeds a memory for sequences that increases the accuracy of prediction for time series forecasting.

— Input gate (i_t): decides which values from the input are used to update the memory.

— Forget gate (f_t): handles what information to throw away from the block.

— Output gate (o_t): manages outputs based on the input and memory gates.

During the training step, each iteration provides an update of the model weights proportional to the partial derivative of the loss. In some cases, the gradient of the loss may be vanishingly small and cause some problems during the learning task such as slow convergence, or even incomplete training if some weights may not be updated. The LSTM architecture has been designed to address the vanishing gradient problem with the use of a gating mechanism described in Figure 9.4.

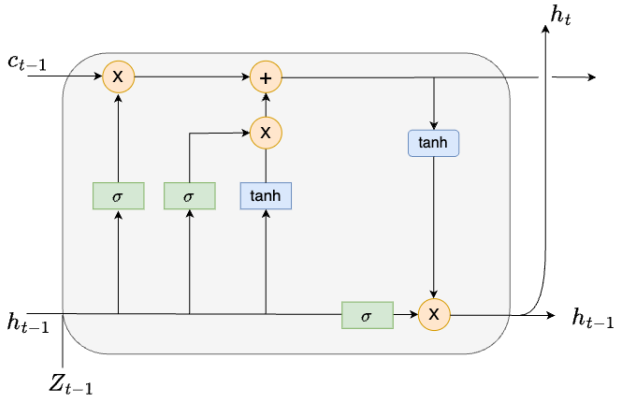


Figure 9.4 – LSTM Cell

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, z_{t-1}] + b_f), \\
i_t &= \sigma(W_i[h_{t-1}, z_{t-1}] + b_i), \\
\tilde{c}_t &= \tanh(W_c[h_{t-1}, z_{t-1}] + b_c), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \\
o_t &= \sigma(W_o[h_{t-1}, z_{t-1}] + b_o), \\
h_t &= o_t \odot \tanh(c_t),
\end{aligned}
\tag{9.11}
$$

where $\Theta := \{W_f, W_i, W_c, W_o\}$ is the set of model parameters and z_{t-1} is the input vector at time t containing all the past information. The number of parameters of each layer in the model should be carefully selected to handle overfitting or underfitting situations.

Attention Free Mechanism

The transformer architecture has made it possible to develop new models capable of being trained on large datasets while being more performant than recurrent neural networks such as LSTM. The Attention Free Transformer (AFT) Zhai et al. [Zha+21a] introduces the attention free block, an alternative to the attention block Vaswani et al. [Vas+17], which eliminates the

need for "dot product self attention" detailed (8.5)-(8.8). The transformer is an efficient tool to capture long term dependencies. Just like transformers, AF Block includes interactions between queries, keys and values. The difference is that the AF block firstly combines key and value together with a set of learned position biases described in Zhai et al. [Zha+21a]. Then the query is combined with the context vector. Let Q, K and V denote the query, key, and value respectively.

$$\begin{aligned} Y &= f(Z), \\ Y_t &= \sigma_q(Q_t) \odot \frac{\sum_{t=1}^{T} exp(K_t + w_t) \odot V_t}{\sum_{t=1}^{T} exp(K_t + w_t)}, \end{aligned} \tag{9.12}$$

we could also rewrite it

$$Y_t = \sigma_q(Q_t) \odot \sum_{t=1}^{T} \big(\text{Sofmax}(K) \odot V\big)_t, \tag{9.13}$$

where $Q = ZW_q$, $K = ZW_k$ and $V = ZW_v$. Where $Z := (Z_0, ...Z_{t-1})$ The activation function σ_q is the sigmoid function and $w_t \in \mathbb{R}^{T \times T}$ is a learned matrix of pair-wise position biases. In the first step, for each target position t, the AFT performs a weighted average of values. In a second step, the result is combined with the query using element-wise multiplication.

Attention Free LSTM

The attention mechanism Vaswani et al. [Vas+17] allows the encoder-decoder models based to be trained only on specific parts of a data sequence that will contribute positively to the prediction of an output. In our case, some part of the past observations will be kept in order to predict the output of our Conditional Autoencoder. Gu, Kelly, and Xiu [GKX21] uses only linear layers to encode and decode the input sequence without necessarily considering temporality and memory. The Attention Free LSTM Layer Inzirillo and De Villelongue [ID22b] brings the possibility of taking into account only the necessary inputs (in addition to the memory effect), and to filter such inputs in order to predict outputs. The AF-LSTM Layer embeds an attention mechanism as well as an activation function that will eliminate some part of the input sequence that is not useful for the reconstruction of this sequence.

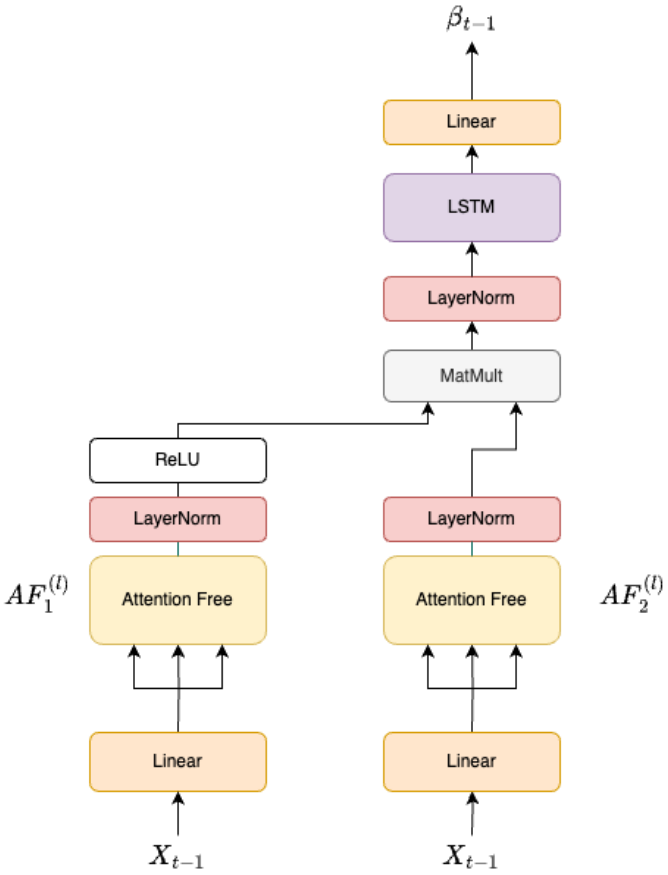


Figure 9.5 – Attention Free LSTM Layer [ID22b]

Our input tensor for each layer, denoted as X, will be filtered on the left side Figure 9.5 using an attention mechanism and an activation function (ReLU) to only propagate the relevant information for prediction through the other hidden layers denoted l. This can be mathematically written as a mapping

$$\tilde{X}_{i,t-1}^{(l)} = AF_1^{(l)}(W_{i,1}^{(l)}; X_{i,t-1}), \quad i \in \{1, ..., N\}, \tag{9.14}$$

where $X_t \in \mathbb{R}^N$ and $AF_{i,1}^{(l)}$ is the Attention Free of the $l-th$ layer for the $i-th$ time series. The latter mapping is defined as follows:

$$
\begin{aligned}
x_{i,t-1}^{(l)} &= \text{Concat}(W_x^{(l)} X_{i,t-1} + b_x^{(l)}), \\
Q_{i,t-1}^{(l)} &= W_q^{(l)} x_{i,t-1}^{(l)}, \\
K_{i,t-1}^{(l)} &= W_k^{(l)} x_{i,t-1}^{(l)}, \\
V_{i,t-1}^{(l)} &= W_v^{(l)} x_{i,t-1}^{(l)}, \\
\eta_{i,t-1}^{(l)} &= \sum_{t'=1}^{s} \text{Softmax}(K_{t-1,t'}^{i,(l)}) \odot V_{t-1,t'}^{i,(l)}, \\
\tilde{X}_{i,t-1}^{(l)} &= \sigma_q(Q_{i,t-1}^{(l)}) \odot \eta_{i,t-1}^{(l)}.
\end{aligned}
\tag{9.15}
$$

We will apply the Layer Normalization (LN) function Ba, Kiros, and Hinton [BKH16], that was developed to compensate the summed input of recurrent neurons:

$$
LN(x; \psi; \phi) = \psi \frac{(x - \mu_x)}{\sigma_x} + \phi,
$$

Moreover, we filter our $\tilde{X}_{t-1}$ using the ReLU activation function, $ReLU(x) = max(0, x)$. Hence we have

$$
\tilde{x}_{t-1} = ReLU(LayerNorm(\tilde{X}_{t-1})), \tag{9.16}
$$

$$
\begin{aligned}
\bar{X}_{i,t-1} &= AF_{i,2}^{(l)}(W_{i,2}^{(l)}; X_{t-1}), \\
\bar{x}_{i,t-1} &= LN(\bar{X}_{t-1}; \psi^{(l)}; \phi^{(l)}).
\end{aligned}
\tag{9.17}
$$

The output from the two channels will be multiplied to apply the filter on our input sequence.

$$
\begin{aligned}
\eta_{i,t-1}^{(l)} &= (\tilde{X}_{i,t-1} \odot \bar{x}_{i,t-1}), \\
\zeta_{i,t-1}^{(l)} &= LN(\eta^{(l)}; \psi^{(l)}; \phi^{(l)}).
\end{aligned}
\tag{9.18}
$$

$\zeta_{t-1}^{(l)}$ will be the input of a parametrized function $g_w^{(l)}$ such that $h_{i,t-1}^{(l)} = g_w^{(l)}(\zeta_{i,t-1}^{(l)})$. In Figure 9.5 $g_w^{(l)}(.)$ is a LSTM (see Figure 9.4). Using the output of $g_w^{(l)}(.)$, we can estimate the output of the $l - th$ layer $\beta_{i,t-1}^{(l)}$ given by

$$
\beta_{i,t-1}^{(l_\beta)} = W_y^{(l_\beta)} h_{i,t-1}^{(l_\beta)} + b_y^{(l_\beta)}, \quad l \in \{1, ..., l_\beta\}. \tag{9.19}
$$

4 Learning task

4.1 Datasets

We record one year of hourly data to perform our analysis on 20 cryptocurrencies. We retrieve the OHLC and volume from CryptoCompare and calculate 24 hours five indicators with the Pandas TA : the Simple Moving Average (SMA) (calculated over 24 hours), the 12-hour Double Exponential Moving Average (DEMA), the Commodity Channel Index (CCI), Accumulation Distribution (AD), and Average True Range (ATR). We then add the time varying Hurst expo-

nent Alvarez-Ramirez et al. [Alv+08] using Detrended Fluctuation Analysis (DFA) [Pen+94]. The process begins by constructing an integrated time series $x(i)$ from the original series $y(i)$, with i ranging from 1 to M, where M denotes the sampling period Δt. This is achieved by summing up the deviations of $y(i)$ from its mean $\bar{y}$. The integrated series is then divided into boxes of equal size m, within which a local trend $z(i;m)$ is determined. The next step involves calculating the fluctuation function $F(m)$ which measures the root mean square deviation of $x(i)$ from the local trend $z(i;m)$. This procedure is repeated across a broad range of box sizes m to assess the scaling behavior of the fluctuation function $F(r)$, with $r = m\Delta t$, which, if following a power-law, suggests the presence of long-range correlations characterized by the Hurst exponent H. Let us define the integrated time series $x(i) = \sum_{j=1}^{i}[y(j) - \bar{y}$ and $z(i;m)$ the local trend in each "box". The fluctuation function is given by

$$F(m) = \sqrt{\frac{1}{M}\sum_{i=1}^{M}[x(i) - z(i;m)]^2}.$$

Repeat the above procedure for a broad range of box sizes m. When the signal follows a scaling law within a given scale domain $m \in D$, a power-law behavior for the fluctuation function $F(\tau)$, with $\tau = m\Delta t$, is observed: $F(\tau) \sim \tau^{\alpha}$, where α is called the scaling or Hurst exponent, a self-affinity parameter representing the long-range correlation properties of the signal. The lower and upper scale limits $m_{\min} \approx 5$ and $m_{\max} \approx M/5$ were recommended by Liu, Gopikrishnan, Stanley, et al. [LGS+99]. For box sizes smaller than $m_{\min} \approx 5$ the fluctuation function $F(\tau)$ carries out deterministic components induced by sampling effects. Finally, we compute the log returns on which we will train our models. In our models, factor loadings β_{t-1} corresponds to the 5 factors values and f_t denotes corresponding factors at time t and refers to the hurst exponent evolution for each timestep. For the learning task, we divided our sample into three temporal sub-periods:

— the first sample is used for training the model, subject to a specific set of hyperparameter values. It corresponds to 70 percent of 1 year of hourly data.

— the second sample is used for validation purpose and the tuning of the hyperparameters. It corresponds to 15 percent of 1 year of hourly data, beginning after the training set.

— the third and last sub-sample is the testing set used to evaluate a model. It corresponds to the last 15 percent of 1 year of hourly data.

4.2 Autoencoders Types

In our study, we have compared three autoencoder architectures. The first model used is a simple autoencoder using linear layers for both encoding and decoding parts. The second model is using an LSTM Autoencoder, using LSTM layers for encoding and decoding. The innovation lies in the third model. The Attention Free LSTM Autoencoder, which incorporates the AF-LSTM layers 9.5. For all thsese three models, we have specific configurations for the inputs and outputs related to two networks, the Beta Network and the Factor Network. The

Beta Network takes five factors from X1 as input, while the Factor Network processes input from twenty selected cryptocurrencies.

4.3 Regularization

To prevent overfitting, several regularization techniques are implemented. We introduced a modification to the objective function by adding a penalty term. Specifically, we apply the LASSO or "l1" penalty. We also used an early stopping mechanism to prevent overfitting during the learning task. The early stopping mechanism is triggered with a tolerance level of 10. This means if the model's performance on the validation set does not improve by a significant margin (defined by this tolerance), the training process will stop. This approach allows us to prevent overfitting and to ensure that the model remains generalized and not "perfectly" fitted to the training set.

4.4 Optimization

To reduce the computational intensity of neural networks optimization in computing the loss function, the following algorithms can be added inside the autoencoders models:

— a stochastic gradient descent (SGD) with an Adam extension to train a neural network that sacrifices accuracy to accelerate the optimization routine. A critical tuning parameter in SGD is the learning rate, which is set to 0.001 for the three types of autoencoders tested.The number of epochs in our case is set to 200.

— the batch normalization that performs one activation over one batch on 20 batches.

— the layer normalization that normalizes the activations of the previous layer for each given example in a batch independently, rather than across a batch like batch Normalization. It is very effective at stabilizing the hidden state dynamics in recurrent networks. It takes the hidden size of the model as input.

4.5 Anomaly identification

The conditional autoencoders are trained on large data sets. The objective is to minimize the mean squared error (MSE) between the input and its reconstruction. The training process involves adjusting the weights of the network to capture the underlying patterns and dependencies in the training data, allowing for an efficient data representation. We assume that our model is capable of estimating the factors and that its explanatory power is such that the little information it fails to explain through the determined factors indicates the presence of an anomaly or several anomalies. To identify these anomalies in terms of log returns, we collect the residuals obtained for the training and test sets. The extreme values at 1 and 5 percent levels are then saved and compared to the log returns observed at the same time. We then consider the log return as abnormal and subject to a market anomaly. We then consider a log return as abnormal when its residual is "extreme" compared to the empirical distribution of such residuals. To determine the threshold for anomaly detection, the reconstruction errors are subjected to a

quantile analysis. Setting the threshold at the 95th percentile of the error distribution implies that instances with errors in the top 5% are classified as anomalies. The log return is considered abnormal if $\hat{r}_t - r_t < q_5$ or $\hat{r}_t - r_t > q_{95}$ and if $r_t > q_{05}$ or $r_t < q_{95}$.

5 Results

In this section, we will analyze the results for Bitcoin and Ethereum only, which are the two largest cryptocurrencies in terms of market capitalization. Additional results will be available in the appendix.

5.1 Bitcoin Anomalies Detection

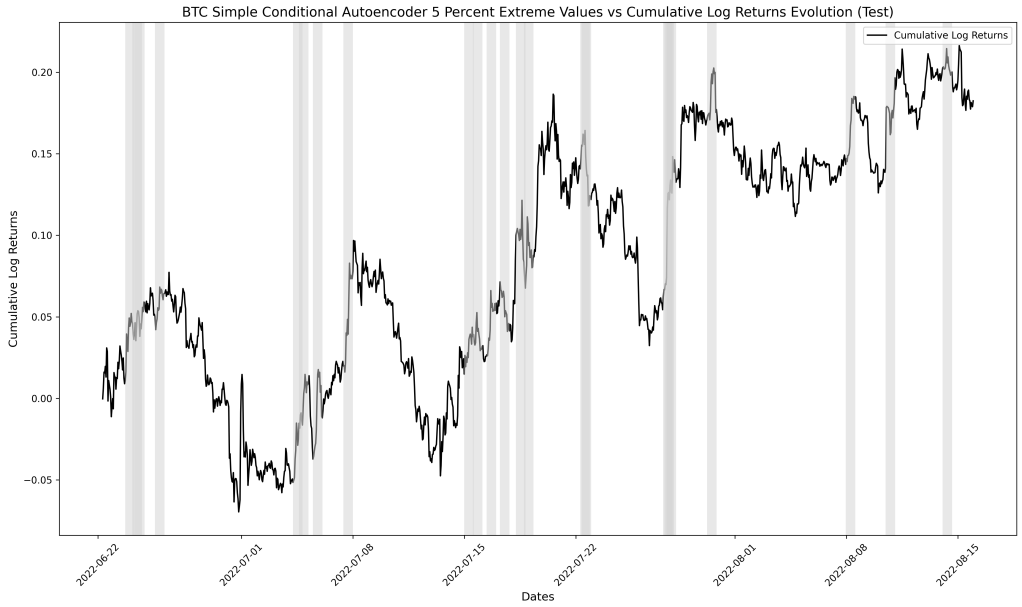


Figure 9.6 – BTC Simple Conditional Autoencoder 5 Percent Extreme Values Evolution on Test Set

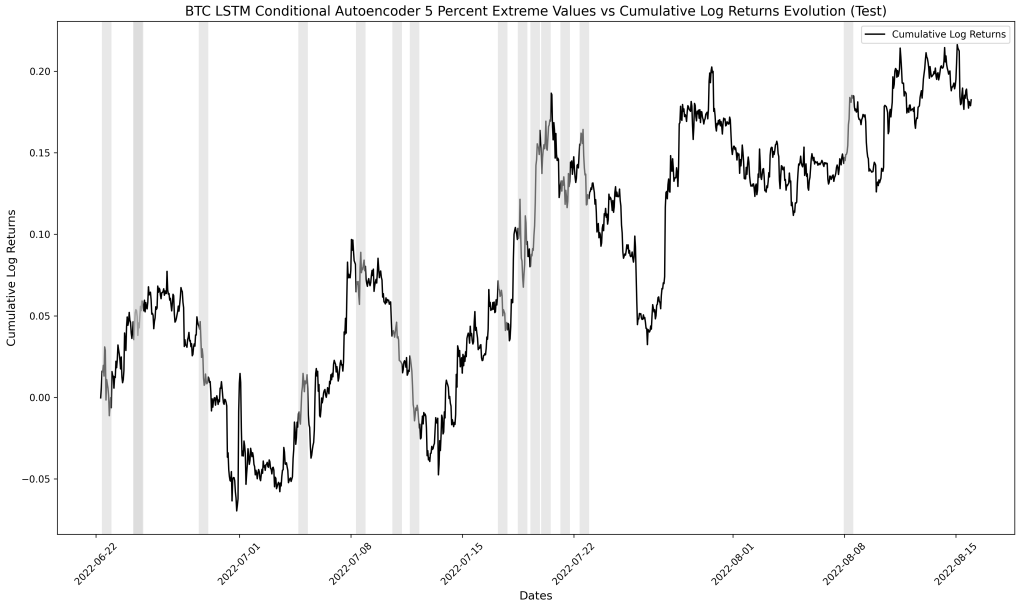


Figure 9.7 – BTC LSTM Conditional autoencoder 5 Percent Extreme Values Evolution on Test Set

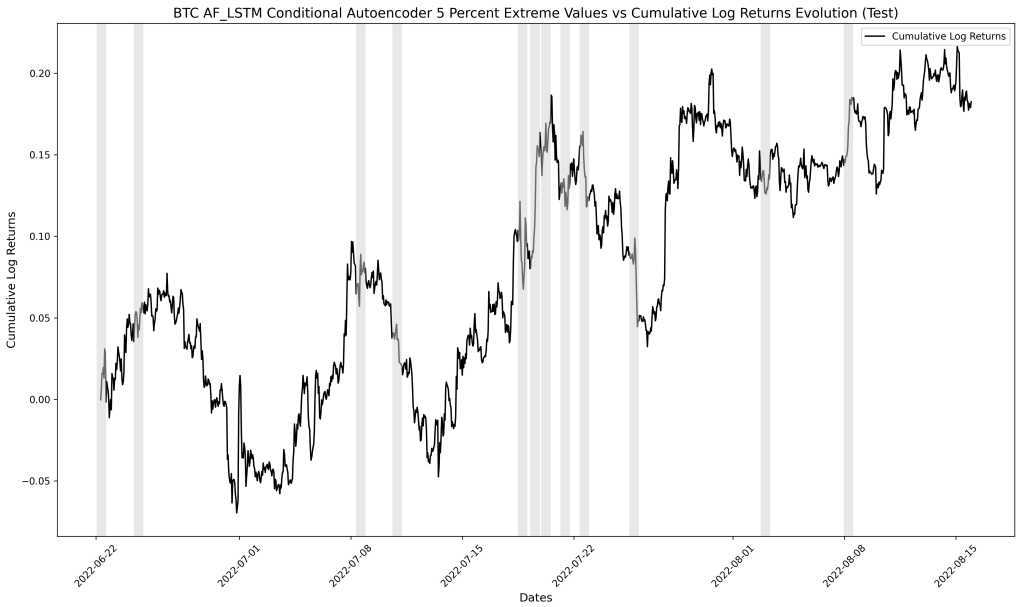


Figure 9.8 – BTC AF-LSTM Conditional Autoencoder 5 Percent Extreme Values Evolution on Test Set

Figures 9.6 9.7- 9.8 show the results for three different models used for Bitcoin anomaly detection: Simple Conditional Autoencoder, LSTM Conditional Autoencoder, and AF-LSTM Conditional Autoencoder. The simplest model, the Linear Conditional Autoencoder, Figure 9.6, shows a relatively high number of detected anomalies, as indicated by the extensive grey shaded areas. However, as more advanced models are used, the number of anomalies detected decreases significantly. The LSTM Conditional Autoencoder has higher forecasting capabilities, reducing

the number of anomalies compared to the initial model. Taking the most sophisticated model, the AF-LSTM Conditional Autoencoder, which incorporates attention mechanisms, exhibits the best performance, with the fewest detected anomalies. These results suggest that the incorporation of advanced deep learning techniques, such as long short-term memory and attention-based models, can enhance the ability to capture the underlying dynamics of Bitcoin price movements and more accurately identify potential anomalies.

6 Conclusion

Although unsupervised techniques are powerful for detecting outliers, they are subject to overfitting and their results might be unstable. Training multiple models to aggregate their scores can be useful to detect how a change in the network structure can change the algorithm precision in terms of predictions or anomaly detection. Outlier detection is a by-product of dimension reduction, explaining the use of autoencoders in our models. The autoencoder techniques can perform nonlinear transformations with their nonlinear activation function and multiple layers. Instead of providing labels that classify the input features, we compare the prediction of the Autoencoder with the initial input features. It turns out that the AF-CAE has a higher explanatory power than the classical autoencoder and thus allows us to better identify anomalies for very noisy time series. In a future work, it would be interesting to look at the detection of anomalies in different market regimes, bullish as well as bearish, knowing that digital assets have for main characteristic to be highly volatile assets.

7 Appendix

7.1 Ethereum Anomalies Detection 95

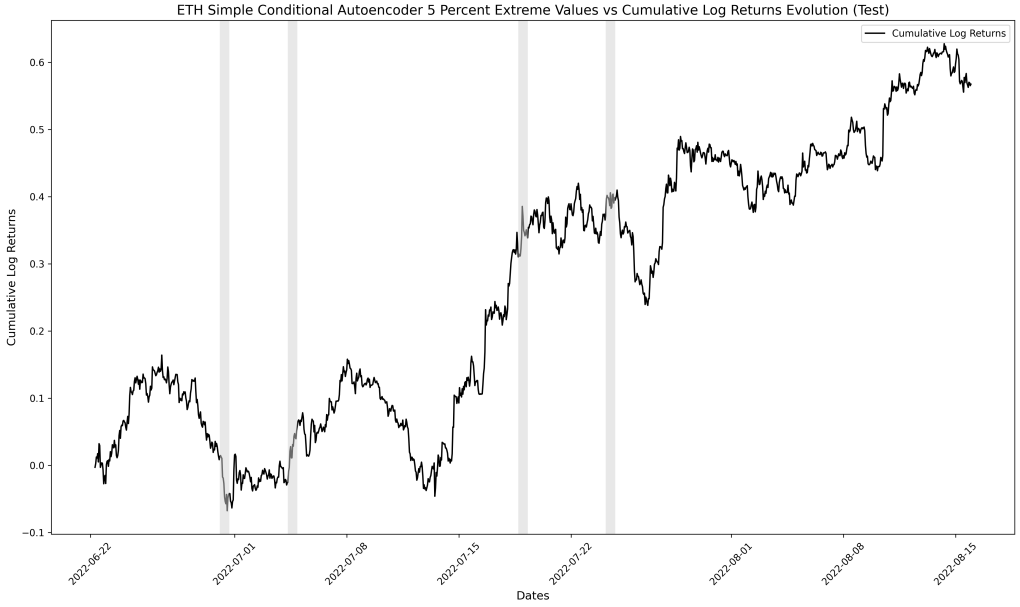


Figure 9.9 – ETH Simple Conditional Autoencoder 5 Percent Extreme Values Evolution on Test Set

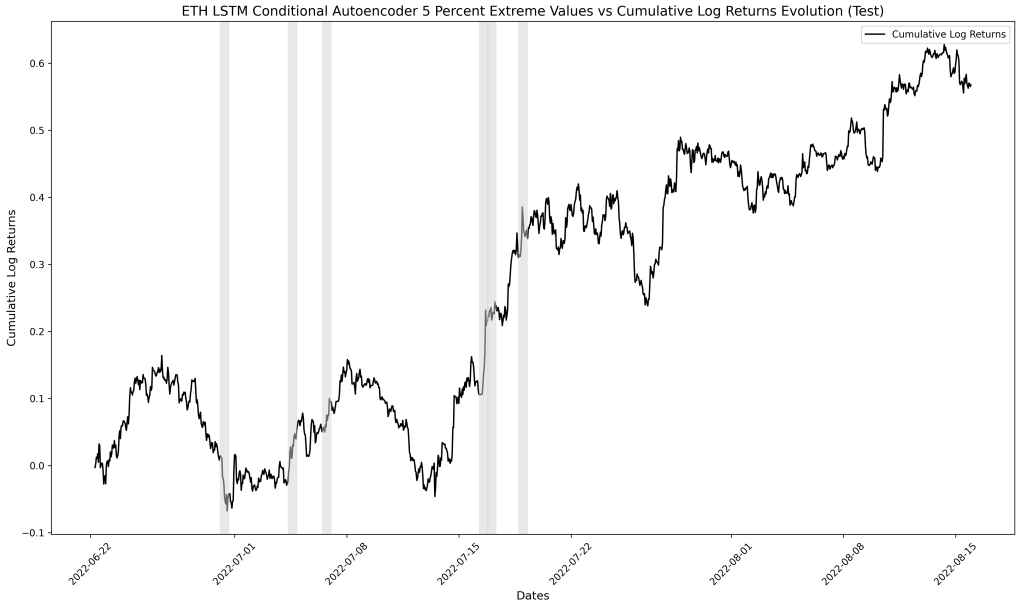


Figure 9.10 – ETH LSTM Conditional Autoencoder 5 Percent Extreme Values Evolution on Test Set

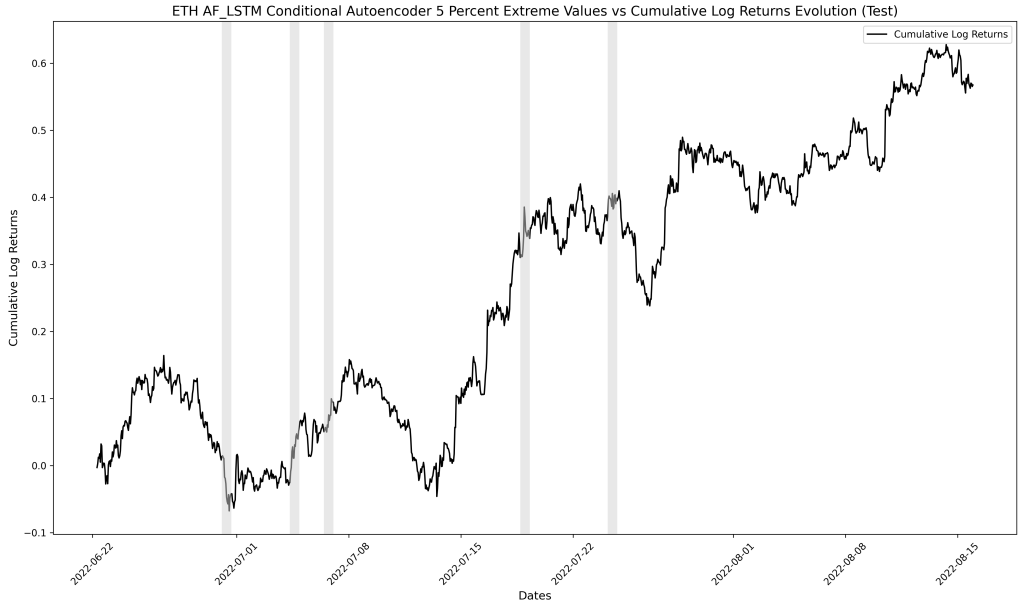


Figure 9.11 – ETH AF-LSTM Conditional Autoencoder 5 Percent Extreme Values Evolution on Test Set

7.2 Ethereum Anomalies Detection 99

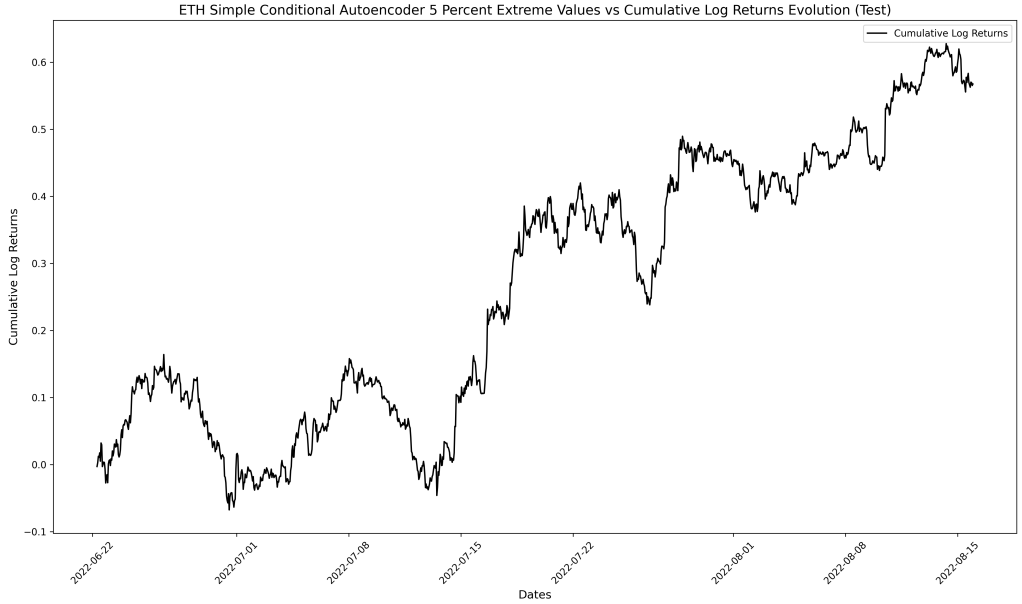


Figure 9.12 – ETH Simple Conditional Autoencoder 1 Percent Extreme Values Evolution on Test Set

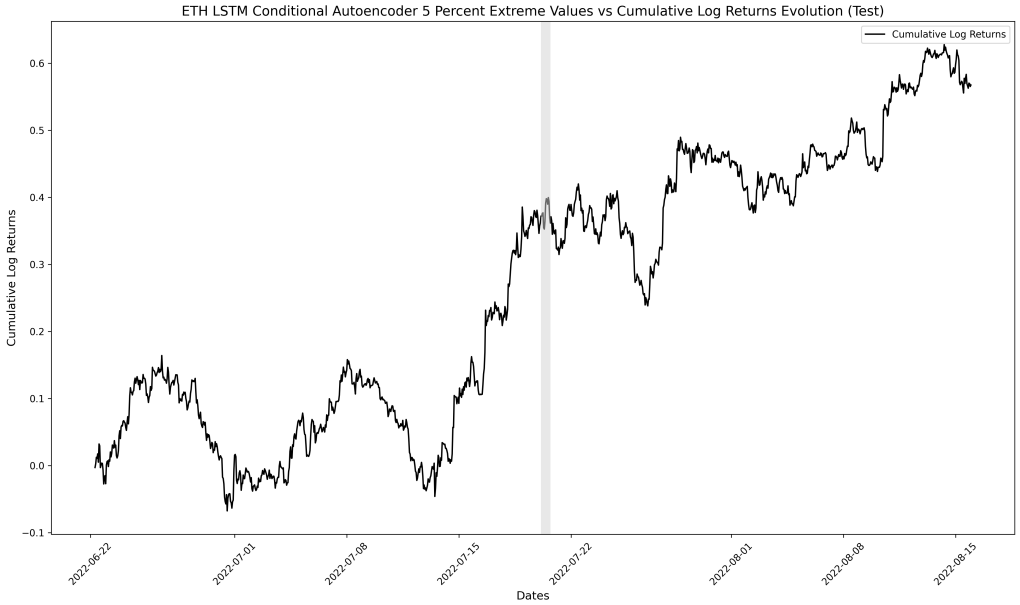


Figure 9.13 – ETH LSTM Conditional Autoencoder 1 Percent Extreme Values Evolution on Test Set

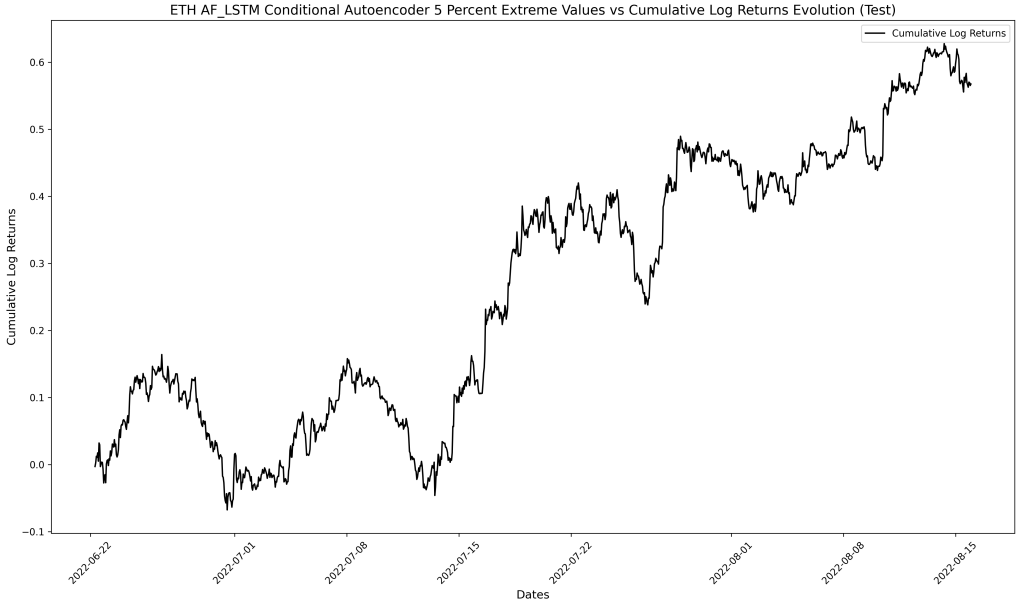


Figure 9.14 – ETH AF-LSTM Conditional Autoencoder 1 Percent Extreme Values Evolution on Test Set

7.3 Bitcoin Anomalies Detection 99

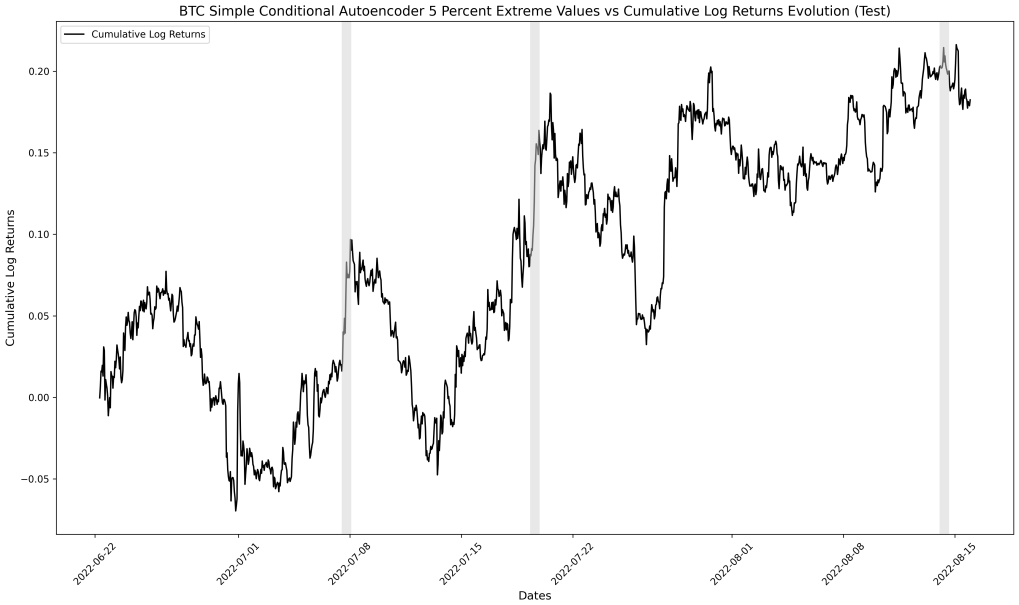


Figure 9.15 – BTC Simple Conditional Autoencoder 1 Percent Extreme Values Evolution on Test Set

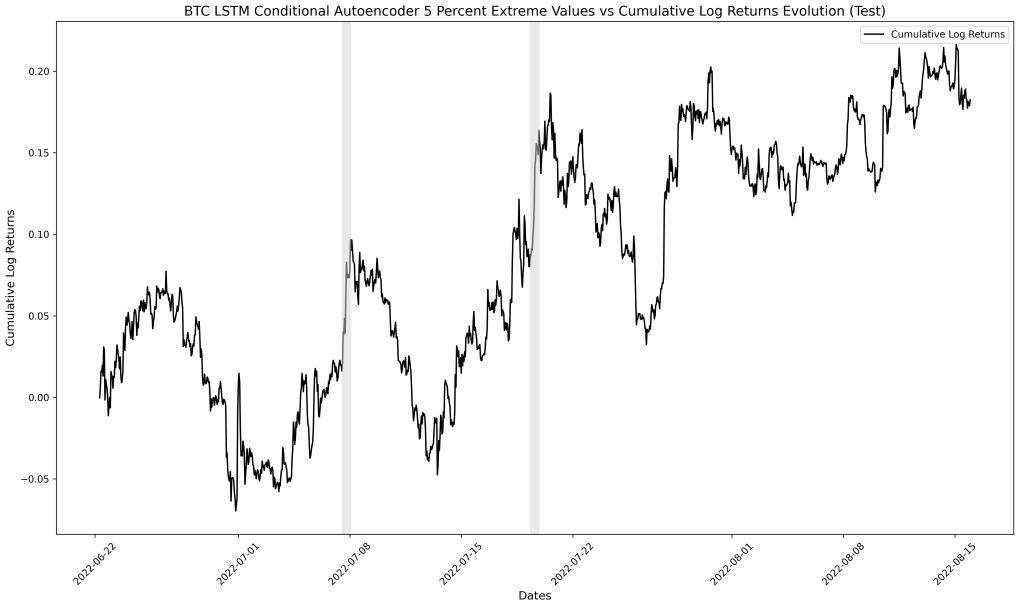


Figure 9.16 – BTC LSTM Conditional Autoencoder 1 Percent Extreme Values Evolution on Test Set

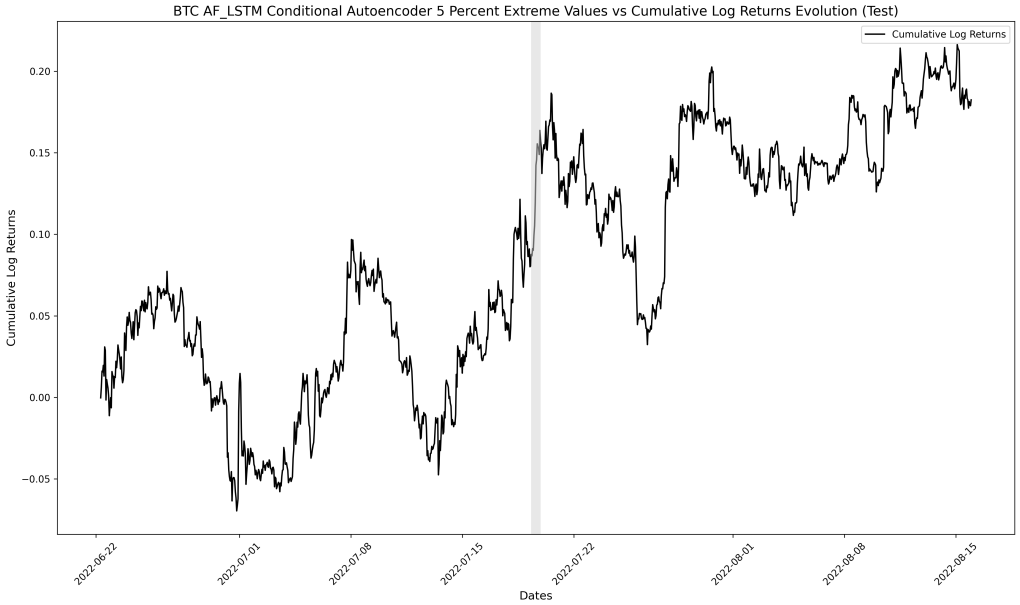


Figure 9.17 – BTC AF-LSTM Conditional Autoencoder 1 Percent Extreme Values Evolution on Test Set

Bibliography

[AMV19] Halvor Aarhus Aalborg, Peter Molnár, and Jon Erik de Vries. « What can explain the price, volatility and trading volume of Bitcoin? » In: *Finance Research Letters* 29 (2019), pp. 255–265.

[Abr+18] Jethin Abraham et al. « Cryptocurrency price prediction using tweet volumes and sentiment analysis ». In: *SMU Data Science Review* 1.3 (2018), p. 1 (cit. on p. 170).

[AS19] Ahmed M. Alaa and Mihaela van der Schaar. « Attentive State-Space Modeling of Disease Progression ». In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019 (cit. on p. 64).

[AM09] Filippo Altissimo and Antonio Mele. « Simulated non-parametric estimation of dynamic models ». In: *The Review of Economic Studies* 76.2 (2009), pp. 413–450 (cit. on p. 112).

[Alv+08] Jose Alvarez-Ramirez et al. « Time-varying Hurst exponent for US stock markets ». In: *Physica A: statistical mechanics and its applications* 387.24 (2008), pp. 6159–6169 (cit. on p. 192).

[And70] Erling Bernhard Andersen. « Asymptotic properties of conditional maximum-likelihood estimators ». In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 32.2 (1970), pp. 283–301 (cit. on p. 64).

[Ant23] Lennart Ante. « How Elon Musk's twitter activity moves cryptocurrency markets ». In: *Technological Forecasting and Social Change* 186 (2023), p. 122112 (cit. on p. 170).

[ABR19] David Ardia, Keven Bluteau, and Maxime Rüede. « Regime changes in Bitcoin GARCH volatility dynamics ». In: *Finance Research Letters* 29 (2019), pp. 266–271 (cit. on pp. 20, 100).

[Ard+18] David Ardia et al. « Forecasting risk with Markov-switching GARCH models: A large-scale performance study ». In: *International Journal of Forecasting* 34.4 (2018), pp. 733–747 (cit. on p. 100).

[Ard+19] David Ardia et al. « Markov-switching GARCH models in R: The MSGARCH package ». In: *Journal of Statistical Software* 91 (2019), pp. 1–38.

[Ark+19] Mikhail Arkhipov et al. « Tuning multilingual transformers for language-specific named entity recognition ». In: *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*. 2019, pp. 89–93 (cit. on p. 46).

[Aug14] Maciej Augustyniak. « Maximum likelihood estimation of the Markov-switching GARCH model ». In: *Computational Statistics & Data Analysis* 76 (2014), pp. 61–75 (cit. on p. 100).

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML] (cit. on pp. 49, 155, 176, 191).

[BCB16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[Bal+17] Mehmet Balcilar et al. « Can volume predict Bitcoin returns and volatility? A quantiles-based approach ». In: *Economic Modelling* 64 (2017), pp. 74–81 (cit. on p. 126).

[BH89] Pierre Baldi and Kurt Hornik. « Neural networks and principal component analysis: Learning from examples without local minima ». In: *Neural networks* 2.1 (1989), pp. 53–58 (cit. on p. 185).

[Bal87] Dana H. Ballard. « Modular Learning in Neural Networks ». In: *Proceedings of the Sixth National Conference on Artificial Intelligence*. Ed. by K. Forbus and H. Shrobe. San Francisco, CA: Morgan Kaufmann, 1987, pp. 279–284.

[Bau+72] Leonard E Baum et al. « An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes ». In: *Inequalities* 3.1 (1972), pp. 1–8.

[BD17] Dirk G Baur and Thomas Dimpfl. « Realized bitcoin volatility ». In: *SSRN* 2949754 (2017), pp. 1–26 (cit. on pp. 15, 126).

[BHH22] Dirk G Baur, Lai T Hoang, and Md Zakir Hossain. « Is Bitcoin a hedge? How extreme volatility can destroy the hedge property ». In: *Finance Research Letters* 47 (2022), p. 102655 (cit. on p. 127).

[BDR14] Luc Bauwens, Arnaud Dufays, and Jeroen VK Rombouts. « Marginal likelihood for Markov-switching and change-point GARCH models ». In: *Journal of Econometrics* 178 (2014), pp. 508–522.

[BLR06] Luc Bauwens, Sébastien Laurent, and Jeroen VK Rombouts. « Multivariate GARCH models: a survey ». In: *Journal of applied econometrics* 21.1 (2006), pp. 79–109 (cit. on p. 102).

[BPR10] Luc Bauwens, Arie Preminger, and Jeroen VK Rombouts. « Theory and inference for a Markov switching GARCH model ». In: *The Econometrics Journal* 13.2 (2010), pp. 218–244 (cit. on pp. 10, 100).

[Bel+19] Irwan Bello et al. *Attention Augmented Convolutional Networks*. 2019. DOI: 10.48550/ARXIV.1904.09925.

[BSF94] Y. Bengio, P. Simard, and P. Frasconi. « Learning long-term dependencies with gradient descent is difficult ». In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181 (cit. on pp. 30, 76).

[BMD18] Mikolaj Binkowski, Gautier Marti, and Philippe Donnat. « Autoregressive convolutional neural networks for asynchronous time series ». In: *International Conference on Machine Learning*. PMLR. 2018, pp. 580–589 (cit. on pp. 5, 44, 151).

[Bit08] Nakamoto S Bitcoin. *Bitcoin: A peer-to-peer electronic cash system*. 2008 (cit. on pp. 9, 20).

[Boe+16] Horatio Boedihardjo et al. « The signature of a rough path: uniqueness ». In: *Advances in Mathematics* 293 (2016), pp. 720–737 (cit. on p. 129).

[Bol86] Tim Bollerslev. « Generalized autoregressive conditional heteroskedasticity ». In: *Journal of Econometrics* 31.3 (1986), pp. 307–327. ISSN: 0304-4076 (cit. on p. 171).

[BLM01] Giovanni Bonanno, Fabrizio Lillo, and Rosario N Mantegna. « High-frequency cross-correlation in a set of stocks ». In: (2001) (cit. on p. 126).

[Bon+03] Giovanni Bonanno et al. « Topology of correlation-based minimal spanning trees in real and model markets ». In: *Physical Review E* 68.4 (2003), p. 046130 (cit. on p. 126).

[BC24] Zavareh Bozorgasl and Hao Chen. « Wav-KAN: Wavelet Kolmogorov-Arnold Networks ». In: *arXiv preprint arXiv:2405.12832* (2024) (cit. on pp. 45, 151).

[BC21] Mateusz Buczyński and Marcin Chlebus. « GARCHNET -VALUE-AT-RISK FORECASTING WITH NOVEL APPROACH TO GARCH MODELS BASED ON NEURAL NETWORKS ». In: (2021).

[BM+16] Jaroslav Bukovina, Matúš Marticek, et al. « Sentiment and bitcoin volatility ». In: *University of Brno* (2016) (cit. on pp. 15, 126).

[Cai94] Jun Cai. « A Markov model of unconditional variance in ARCH ». In: *Journal of Business and Economic Statistics* 12.3 (1994), pp. 309–316 (cit. on p. 120).

[CZ19] Guglielmo Maria Caporale and Timur Zekokh. « Modelling volatility of cryptocurrencies using Markov-Switching GARCH models ». In: *Research in International Business and Finance* 48 (2019), pp. 143–155 (cit. on p. 100).

[CD19] Amélie Charles and Olivier Darné. « The accuracy of asymmetric GARCH model estimation ». In: *International Economics* 157 (2019), pp. 179–202. ISSN: 2110-7017 (cit. on p. 77).

[CF15] Eng-Tuck Cheah and John Fry. « Speculative bubbles in Bitcoin markets? An empirical investigation into the fundamental value of Bitcoin ». In: *Economics Letters* 130 (2015), pp. 32–36. ISSN: 0165-1765 (cit. on p. 64).

[Che58] Kuo-Tsai Chen. « Integration of paths–A faithful representation of paths by noncommutative formal power series ». In: *Transactions of the American Mathematical Society* 89.2 (1958), pp. 395–407 (cit. on pp. 15, 126, 127, 151).

[CK16] Ilya Chevyrev and Andrey Kormilitzin. « A primer on the signature method in machine learning ». In: *arXiv preprint arXiv:1603.03788* (2016) (cit. on pp. 15, 127–129, 151).

[Chi+19] Rewon Child et al. « Generating long sequences with sparse transformers ». In: *arXiv preprint arXiv:1904.10509* (2019).

[Cho+14a] KyungHyun Cho et al. « On the Properties of Neural Machine Translation: Encoder-Decoder Approaches ». In: *CoRR* abs/1409.1259 (2014) (cit. on p. 75).

[Cho+14b] Kyunghyun Cho et al. « Learning phrase representations using RNN encoder-decoder for statistical machine translation ». In: *arXiv preprint arXiv:1406.1078* (2014).

[CC08] Yves Choueifaty and Yves Coignard. « Toward maximum diversification ». In: *The Journal of Portfolio Management* 35.1 (2008), pp. 40–51 (cit. on pp. 126, 135).

[CT12] Yuan Shih Chow and Henry Teicher. *Probability theory: independence, interchangeability, martingales*. Springer Science & Business Media, 2012 (cit. on p. 104).

[Chu+17] Jeffrey Chu et al. « GARCH modelling of cryptocurrencies ». In: *Journal of Risk and Financial Management* 10.4 (2017), p. 17.

[Chu+14] Junyoung Chung et al. « Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling ». In: *CoRR* abs/1412.3555 (2014) (cit. on pp. 30, 76).

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. « Fast and accurate deep network learning by exponential linear units (elus) ». In: *arXiv preprint arXiv:1511.07289* (2015) (cit. on pp. 49, 155).

[CUH16] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. « Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs) ». In: (2016).

[CK12] Michael Creel and Dennis Kristensen. « Estimation of dynamic latent variable models using simulated non-parametric moments ». In: *The Econometrics Journal* 15.3 (2012), pp. 490–515 (cit. on p. 112).

[CF20] Alessandra Cretarola and Gianna Figà-Talamanca. « Bubble regime identification in an attention-based model for Bitcoin and Ethereum price dynamics ». In: *Economics Letters* 191 (2020), p. 108831. ISSN: 0165-1765 (cit. on p. 64).

[CF21] Alessandra Cretarola and Gianna Figà-Talamanca. « Detecting bubbles in Bitcoin price dynamics via market exuberance ». In: *Annals of Operations Research* 299 (2021), pp. 459–479 (cit. on p. 64).

[Cyb89] George Cybenko. « Approximation by superpositions of a sigmoidal function ». In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314 (cit. on pp. 2, 30, 31, 75, 152).

[Dai+21] Xiyang Dai et al. « Dynamic detr: End-to-end object detection with dynamic attention ». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 2988–2997.

[Dai+01] Z Dai et al. « Attentive Language Models Beyond a Fixed-Length Context. Published online June 2, 2019. doi: 10.48550 ». In: *arXiv* (1901) (cit. on p. 46).

[Dai+20] Zihang Dai et al. « Funnel-transformer: Filtering out sequential redundancy for efficient language processing ». In: *Advances in neural information processing systems* 33 (2020), pp. 4271–4282.

[Dau+17] Yann N Dauphin et al. « Language modeling with gated convolutional networks ». In: *International conference on machine learning*. PMLR. 2017, pp. 933–941 (cit. on pp. 49, 156).

[DLW93] Francis Diebold, Joon-Haeng Lee, and Gretchen Weinbach. *Regime switching with time-varying transition probabilities*. Tech. rep. Federal Reserve Bank of Philadelphia, 1993 (cit. on pp. 66, 78).

[DD22] Haizhou Du and Ziyi Duan. « Finder: A novel approach of change point detection for multivariate time series ». In: *Applied Intelligence* 52 (2022). DOI: 10.1007/s10489-021-02532-x.

[DCS21] Joel Dyer, Patrick W Cannon, and Sebastian M Schmon. « Deep signature statistics for likelihood-free time-series models ». In: *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*. 2021 (cit. on pp. 15, 128, 151).

[EUD18] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. « Sigmoid-weighted linear units for neural network function approximation in reinforcement learning ». In: *Neural networks* 107 (2018), pp. 3–11 (cit. on p. 155).

[Fan+19] Chenyou Fan et al. « Multi-horizon time series forecasting with temporal attention learning ». In: *Proceedings of the 25th ACM SIGKDD International conference on knowledge discovery & data mining*. 2019, pp. 2527–2535 (cit. on p. 36).

[Fan+21] Zhihao Fan et al. « Mask attention networks: Rethinking and strengthen transformer ». In: *arXiv preprint arXiv:2103.13597* (2021) (cit. on p. 46).

[FWZ18] Wenjun Feng, Yiming Wang, and Zhengjun Zhang. « Can cryptocurrencies be a safe haven: a tail risk perspective analysis ». In: *Applied Economics* 50.44 (2018), pp. 4745–4762.

[Fer21] Adeline Fermanian. « Embedding and learning with signatures ». In: *Computational Statistics & Data Analysis* 157 (2021), p. 107148 (cit. on pp. 15, 128, 151).

[FS04] Jean-David Fermanian and Bernard Salanie. « A nonparametric simulated maximum likelihood estimation method ». In: *Econometric Theory* 20.4 (2004), pp. 701–734 (cit. on pp. 112, 113).

[Fos21] Peter Foster. *A Brief Introduction to Path Signatures*. URL. 2021.

[FZ19] Christian Francq and Jean-Michel Zakoian. *GARCH models: structure, statistical inference and financial applications*. John Wiley & Sons, 2019 (cit. on pp. 102, 103, 106, 108, 172).

[FZ+08] Christian Francq, Jean-Michel Zakoı, et al. « Deriving the autocovariances of powers of Markov-switching GARCH models, with applications to statistical inference ». In: *Computational Statistics & Data Analysis* 52.6 (2008), pp. 3027–3046 (cit. on p. 100).

[Frü06] Sylvia Frühwirth-Schnatter. *Finite mixture and Markov switching models*. Springer, 2006 (cit. on p. 98).

[Gar+14] David Garcia et al. « The digital traces of bubbles: Feedback cycles between socio-economic signals in the Bitcoin economy ». In: *Journal of the Royal Society, Interface / the Royal Society* 11 (2014) (cit. on p. 64).

[GB20] Laura Garcia-Jorcano and Sonia Benito. « Studying the properties of the Bitcoin as a diversifying and hedging asset through a copula analysis: Constant and time-varying ». In: *Research in International Business and Finance* 54 (2020), p. 101300. ISSN: 0275-5319 (cit. on p. 64).

[GI24a] Remi Genet and Hugo Inzirillo. « A Temporal Kolmogorov-Arnold Transformer for Time Series Forecasting ». In: *arXiv preprint arXiv:2406.02486* (2024) (cit. on pp. 151, 158, 160).

[GI24b] Remi Genet and Hugo Inzirillo. « TKAN: Temporal Kolmogorov-Arnold Networks ». In: *arXiv preprint arXiv:2405.07344* (2024) (cit. on pp. 45, 46, 52, 54, 151, 158, 160).

[Gla+14] Florian Glaser et al. « Bitcoin-asset or currency? revealing users' hidden intentions ». In: *Revealing Users' Hidden Intentions (April 15, 2014). ECIS* (2014) (cit. on pp. 9, 20).

[GBB10] Xavier Glorot, Antoine Bordes, and Y. Bengio. « Deep Sparse Rectifier Neural Networks ». In: vol. 15. 2010.

[GJR93] Lawrence R. Glosten, Ravi Jagannathan, and David E. Runkle. « On the Relation between the Expected Value and the Volatility of the Nominal Excess Return on Stocks ». In: *The Journal of Finance* 48.5 (1993), pp. 1779–1801. ISSN: 00221082, 15406261 (cit. on p. 172).

[Gra96] Stephen F Gray. « Modeling the conditional distribution of interest rates as a regime-switching process ». In: *Journal of Financial Economics* 42.1 (1996), pp. 27–62 (cit. on pp. 10, 99, 119, 120).

[GKX21] Shihao Gu, Bryan Kelly, and Dacheng Xiu. « Autoencoder asset pricing models ». In: *Journal of Econometrics* 222.1 (2021), pp. 429–450 (cit. on pp. xi, 23–25, 184–187, 189).

[Gyu+13] Lajos Gergely Gyurkó et al. « Extracting information from the signature of a financial data stream ». In: *arXiv preprint arXiv:1307.7244* (2013) (cit. on pp. 15, 128, 130, 151).

[HMP04] Markus Haas, Stefan Mittnik, and Marc S Paolella. « A new approach to Markov-switching GARCH models ». In: *Journal of financial Econometrics* 2.4 (2004), pp. 493–530 (cit. on pp. 10, 11, 97, 99, 100, 112, 120).

[HL10] Ben Hambly and Terry Lyons. « Uniqueness for the signature of a path of bounded variation and the reduced path group ». In: *Annals of Mathematics* (2010), pp. 109–167 (cit. on p. 129).

[Ham89a] James D Hamilton. « A new approach to the economic analysis of nonstationary time series and the business cycle ». In: *Econometrica: Journal of the econometric society* (1989), pp. 357–384 (cit. on pp. 9, 98, 100).

[Ham94] James D Hamilton. « State-space models ». In: *Handbook of econometrics* 4 (1994), pp. 3039–3080 (cit. on pp. 5, 44, 151).

[HS94] James D Hamilton and Raul Susmel. « Autoregressive conditional heteroskedasticity and changes in regime ». In: *Journal of econometrics* 64.1-2 (1994), pp. 307–333 (cit. on p. 120).

[Ham89b] James D. Hamilton. « A New Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle ». In: *Econometrica* 57.2 (1989), pp. 357–384. ISSN: 00129682, 14680262 (cit. on pp. 64–66, 78).

[Ham20] James Douglas Hamilton. *Time series analysis*. Princeton university press, 2020 (cit. on p. 118).

[Han+22] Kai Han et al. « A survey on vision transformer ». In: *IEEE transactions on pattern analysis and machine intelligence* 45.1 (2022), pp. 87–110.

[HTF09] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN: 9780387848846.

[Hay98] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998 (cit. on pp. 30, 75).

[HBB21] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. « Recurrent neural networks for time series forecasting: Current status and future directions ». In: *International Journal of Forecasting* 37.1 (2021), pp. 388–427 (cit. on p. 30).

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. « Reducing the dimensionality of data with neural networks ». In: *science* 313.5786 (2006), pp. 504–507 (cit. on p. 185).

[Hoc98a] Sepp Hochreiter. « The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions ». In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (1998), pp. 107–116. DOI: 10.1142/S0218488598000094.

[Hoc98b] Sepp Hochreiter. « The vanishing gradient problem during learning recurrent neural nets and problem solutions ». In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116 (cit. on p. 30).

[HS97a] Sepp Hochreiter and Jürgen Schmidhuber. « Long Short-term Memory ». In: *Neural computation* 9 (1997), pp. 1735–80 (cit. on pp. 82, 170, 173, 187).

[HS97b] Sepp Hochreiter and Jürgen Schmidhuber. « Long short-term memory ». In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on pp. 5, 35, 44, 45, 47, 151).

[HJ12] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012 (cit. on p. 110).

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. « Multilayer feedforward networks are universal approximators ». In: *Neural networks* 2.5 (1989), pp. 359–366 (cit. on pp. 30, 31, 75, 152).

[Ilh+20] Fatih Ilhan et al. *Markovian RNN: An Adaptive Time Series Prediction Network with HMM-based Switching for Nonstationary Environments*. 2020.

[Ilh+21] Fatih Ilhan et al. « Markovian RNN: An adaptive time series prediction network with HMM-based switching for nonstationary environments ». In: *IEEE Transactions on Neural Networks and Learning Systems* (2021) (cit. on p. 64).

[ID22a] Hugo Inzirillo and Stanislas De Quénetain. « Managing Risk in DeFi Portfolios ». In: *arXiv preprint arXiv:2205.14699* (2022) (cit. on p. 170).

[ID22b] Hugo Inzirillo and Ludovic De Villelongue. « An Attention Free Long Short-Term Memory for Time Series Forecasting ». In: *arXiv preprint arXiv:2209.09548* (2022) (cit. on pp. 126, 182, 189, 190).

[IM21] Hugo Inzirillo and Benjamin Mat. « Dimensionality reduction for prediction: Application to Bitcoin and Ethereum ». In: *arXiv preprint arXiv:2112.15036* (2021) (cit. on pp. 64, 69).

[Jor85] Philippe Jorion. « International portfolio diversification with estimation risk ». In: *Journal of Business* (1985), pp. 259–278.

[Kal60] Rudolph Emil Kalman. « A new approach to linear filtering and prediction problems ». In: (1960).

[Kat17] Paraskevi Katsiampa. « Volatility estimation for Bitcoin: A comparison of GARCH models ». In: *Economics Letters* 158 (2017), pp. 3–6. ISSN: 0165-1765 (cit. on p. 64).

[KS84] Harry Kesten and Frank Spitzer. « Convergence in distribution of products of random matrices ». In: *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 67 (1984), pp. 363–386 (cit. on p. 109).

[Kim94] Chang-Jin Kim. « Dynamic linear models with Markov-switching ». In: *Journal of econometrics* 60.1-2 (1994), pp. 1–22 (cit. on pp. 5, 44, 151).

[KW18] Ha Young Kim and Chang Hyun Won. « Forecasting the volatility of stock price index: A hybrid model integrating LSTM with multiple GARCH-type models ». In: *Expert Systems with Applications* 103 (2018), pp. 25–37. ISSN: 0957-4174 (cit. on pp. 21, 172).

[Kim+18] Tae Hyun Kim et al. « Spatio-temporal transformer network for video restoration ». In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 106–122.

[Kla02] Franc Klaassen. « Improving GARCH volatility forecasts with regime-switching GARCH ». In: *Advances in Markov-switching models*. Springer, 2002, pp. 223–254 (cit. on pp. 10, 99, 120).

[Kol61] Andrei Nikolaevich Kolmogorov. *On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables*. American Mathematical Society, 1961 (cit. on pp. 2, 31, 46, 151, 152).

[Kor18] Jerzy Korzeniewski. « Efficient stock portfolio construction by means of clustering ». In: (2018) (cit. on p. 127).

[Koz19] Serhiy Kozak. « Kernel Trick for the Cross Section ». In: *SSRN Electronic Journal* (2019). DOI: 10.2139/ssrn.3307895 (cit. on p. 184).

[Kra91] Mark A. Kramer. « Nonlinear principal component analysis using autoassociative neural networks ». In: *AIChE Journal* 37.2 (1991), pp. 233–243.

[KS12] Dennis Kristensen and Yongseok Shin. « Estimation of dynamic models with nonparametric simulated maximum likelihood ». In: *Journal of Econometrics* 167.1 (2012), pp. 76–94 (cit. on pp. 112, 113).

[KM15] Werner Kristjanpoller and Marcel C. Minutolo. « Gold price volatility: A forecasting approach using the Artificial Neural Network–GARCH model ». In: *Expert Systems with Applications* 42.20 (2015), pp. 7245–7251. ISSN: 0957-4174 (cit. on pp. 21, 172).

[Kri15] Ladislav Kristoufek. « What are the main drivers of the Bitcoin price? Evidence from wavelet coherence analysis ». In: *PloS one* 10.4 (2015), e0123923 (cit. on p. 15).

[KGP23] Andromahi Kufo, Ardit Gjeci, and Artemisa Pilkati. « Unveiling the Influencing Factors of Cryptocurrency Return Volatility ». In: *Journal of Risk and Financial Management* 17.1 (2023), p. 12.

[KC18] Che-Yu Kuo and Jen-Tzung Chien. « Markov recurrent neural networks ». In: *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE. 2018, pp. 1–6 (cit. on p. 64).

[LL90] Christopher G Lamoureux and William D Lastrapes. « Heteroskedasticity in stock return data: Volume versus GARCH effects ». In: *The journal of finance* 45.1 (1990), pp. 221–229.

[LJH15] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. « A simple way to initialize recurrent networks of rectified linear units ». In: *arXiv preprint arXiv:1504.00941* (2015) (cit. on pp. 30, 75).

[LKS91] Yann Lecun, Ido Kanter, and Sara Solla. « Eigenvalues of covariance matrices: Application to neural-network learning ». In: *Physical Review Letters* 66 (1991), pp. 2396–2399. DOI: 10.1103/PhysRevLett.66.2396.

[Lec+00] Yann Lecun et al. « Efficient BackProp ». In: (2000).

[Leó+17] Diego León et al. « Clustering algorithms for risk-adjusted portfolio construction ». In: *Procedia Computer Science* 108 (2017), pp. 1334–1343 (cit. on p. 127).

[LLN13] Daniel Levin, Terry Lyons, and Hao Ni. « Learning from the past, predicting the statistics for the future, learning an evolving system ». In: *arXiv preprint arXiv:1309.0260* (2013) (cit. on p. 130).

[LX23] Edmond Lezmi and Jiali Xu. « Time Series Forecasting with Transformer Models and Application to Asset Management ». In: *Available at SSRN 4375798* (2023) (cit. on p. 174).

[LZJ17] Chenyang Li, Xin Zhang, and Lianwen Jin. « LPSNet: a novel log path signature feature based hand gesture recognition framework ». In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 631–639 (cit. on pp. 15, 128, 151).

[Li+19] Shiyang Li et al. « Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting ». In: *Advances in neural information processing systems* 32 (2019) (cit. on pp. 46, 51).

[Li+20] Shiyang Li et al. *Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting*. 2020. arXiv: 1907.00235 [cs.LG].

[Lim+21] Bryan Lim et al. « Temporal fusion transformers for interpretable multi-horizon time series forecasting ». In: *International Journal of Forecasting* 37.4 (2021), pp. 1748–1764 (cit. on pp. 1, 6, 36, 45, 46, 48, 49, 155).

[Lin+22] Tianyang Lin et al. « A survey of transformers ». In: *AI open* 3 (2022), pp. 111–132.

[LS19] Jinan Liu and Apostolos Serletis. « Volatility in the cryptocurrency market ». In: *Open Economies Review* 30 (2019), pp. 779–811.

[Liu+18] Peter J Liu et al. « Generating wikipedia by summarizing long sequences ». In: *arXiv preprint arXiv:1801.10198* (2018).

[LS20] Wing Ki Liu and Mike K. P. So. « A GARCH Model with Artificial Neural Networks ». In: *Information* 11.10 (2020). ISSN: 2078-2489 (cit. on pp. 21, 172).

[LGS+99] Yanhui Liu, Parameswaran Gopikrishnan, H Eugene Stanley, et al. « Statistical properties of the volatility of price fluctuations ». In: *Physical review e* 60.2 (1999), p. 1390 (cit. on p. 192).

[Liu+24] Ziming Liu et al. « KAN: Kolmogorov-Arnold Networks ». In: *arXiv preprint arXiv:2404.19756* (2024) (cit. on pp. 1, 2, 18, 30, 31, 33, 45, 81, 151–153).

[Lon05] Francois Longin. « The choice of the distribution of asset returns: How extreme value theory can help? » In: *Journal of Banking & Finance* 29.4 (2005), pp. 1017–1035.

[LAM22] Carmen López-Martín, Raquel Arguedas-Sanz, and Sonia Benito Muela. « A cryptocurrency empirical study focused on evaluating their distribution functions ». In: *International Review of Economics and Finance* 79 (2022), pp. 387–407. ISSN: 1059-0560 (cit. on p. 64).

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. « Effective approaches to attention-based neural machine translation ». In: *arXiv preprint arXiv:1508.04025* (2015) (cit. on p. 175).

[Lüt97] Helmut Lütkepohl. *Handbook of matrices*. John Wiley & Sons, 1997 (cit. on p. 110).

[Lüt13] Helmut Lütkepohl. « Vector autoregressive models ». In: *Handbook of research methods and applications in empirical macroeconomics*. Edward Elgar Publishing, 2013, pp. 139–164 (cit. on pp. 5, 44, 151).

[Lyo14] Terry Lyons. « Rough paths, signatures and the modelling of functions on streams ». In: *arXiv preprint arXiv:1405.4537* (2014) (cit. on pp. 127, 129, 151).

[Lyo98] Terry J Lyons. « Differential equations driven by rough signals ». In: *Revista Matemática Iberoamericana* 14.2 (1998), pp. 215–310 (cit. on pp. 127, 151).

[ML19] King Ma and Henry Leung. « A novel LSTM approach for asynchronous multivariate time series prediction ». In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–7 (cit. on pp. 5, 44, 151).

[Mac+67] James MacQueen et al. « Some methods for classification and analysis of multivariate observations ». In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[MH97] Spyros Makridakis and Michele Hibon. « ARMA models and the Box–Jenkins methodology ». In: *Journal of forecasting* 16.3 (1997), pp. 147–163 (cit. on p. 30).

[MS06] Yannick Malevergne and Didier Sornette. *Extreme financial risks: From dependence to risk management*. Springer Science & Business Media, 2006 (cit. on p. 22).

[MQ23] The Nguyen Manh and Hoan Bui Quoc. « Portfolio Construction Based on Time Series Clustering Method Evidence in the Vietnamese Stock Market ». In: *International Conference on Computational Data and Social Networks*. Springer. 2023, pp. 129–137 (cit. on p. 131).

[Man99] Rosario N Mantegna. « Hierarchical structure in financial markets ». In: *The European Physical Journal B-Condensed Matter and Complex Systems* 11 (1999), pp. 193–197 (cit. on p. 126).

[MSW06] Massimiliano Marcellino, James H Stock, and Mark W Watson. « A comparison of direct and iterated multistep AR methods for forecasting macroeconomic time series ». In: *Journal of econometrics* 135.1-2 (2006), pp. 499–526 (cit. on pp. 5, 44, 151).

[MT00] Harry M Markowitz and G Peter Todd. *Mean-variance analysis in portfolio choice and capital markets*. Vol. 66. John Wiley & Sons, 2000 (cit. on pp. 126, 135).

[Mar15] Karina Marvin. « Creating diversified portfolios using cluster analysis ». In: *Princeton University* (2015) (cit. on p. 127).

[MP43] Warren Mcculloch and Walter Pitts. « A Logical Calculus of Ideas Immanent in Nervous Activity ». In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 127–147 (cit. on p. 73).

[MJ+01] Larry R Medsker, Lakhmi Jain, et al. « Recurrent neural networks ». In: *Design and Applications* 5.64-67 (2001), p. 2 (cit. on p. 30).

[Meh+20] Sachin Mehta et al. « Delight: Deep and light-weight transformer ». In: *arXiv preprint arXiv:2008.00623* (2020) (cit. on p. 46).

[MSG14] Prapanna Mondal, Labani Shit, and Saptarsi Goswami. « Study of effectiveness of time series modeling (ARIMA) in forecasting stock prices ». In: *International Journal of Computer Science, Engineering and Applications* 4.2 (2014), p. 13 (cit. on p. 30).

[Moo+22] Paul Moore et al. « Path signatures for non-intrusive load monitoring ». In: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2022, pp. 3808–3812.

[NH10a] Vinod Nair and Geoffrey E Hinton. « Rectified linear units improve restricted boltzmann machines ». In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[NH10b] Vinod Nair and Geoffrey E. Hinton. « Rectified Linear Units Improve Restricted Boltzmann Machines ». In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 9781605589077.

[Nak09] Satoshi Nakamoto. « Bitcoin: A Peer-to-Peer Electronic Cash System ». In: (2009) (cit. on p. 64).

[NR16] Hamzaoui Nessrine and Boutheina Regaieg. « The Glosten-Jagannathan-Runkle-Generalized Autoregressive Conditional Heteroscedastic Approach to Investigating the Foreign Exchange Forward Premium Volatility ». In: *International Journal of Economics and Financial Issues* 6 (2016), pp. 1608–1615.

[NTS13] Nikolay Nikolaev, Peter Tino, and Evgueni Smirnov. « Time-dependent series variance learning with recurrent mixture density networks ». In: *Neurocomputing* 122 (2013), pp. 501–512. ISSN: 0925-2312 (cit. on pp. 21, 172).

[Nik+19] Nikolay Y. Nikolaev et al. « A regime-switching recurrent neural network model applied to wind time series ». In: *Applied Soft Computing* (2019) (cit. on p. 64).

[PPS22] Theodore Panagiotidis, Georgios Papapanagiotou, and Thanasis Stengos. « On the volatility of cryptocurrencies ». In: *Research in International Business and Finance* 62 (2022), p. 101724 (cit. on p. 100).

[Pen+94] C-K Peng et al. « Mosaic organization of DNA nucleotides ». In: *Physical review e* 49.2 (1994), p. 1685 (cit. on p. 192).

[Per+18] Imanol Perez Arribas et al. « A signature-based machine learning model for distinguishing bipolar disorder and borderline personality disorder ». In: *Translational psychiatry* 8.1 (2018), p. 274 (cit. on pp. 15, 128, 151).

[PK17] Lukáš Pichl and Taisei Kaizoji. « Volatility analysis of bitcoin ». In: *Quantitative Finance and Economics* 1.4 (2017), pp. 474–485 (cit. on pp. 15, 126).

[Pol+19] Adrian Alan Pol et al. « Anomaly detection with conditional variational autoencoders ». In: *2019 18th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2019, pp. 1651–1657 (cit. on pp. 23, 184).

[Pro18] David Procházka. « Accounting for Bitcoin and Other Cryptocurrencies under IFRS: A Comparison and Assessment of Competing Models ». In: *The International Journal of Digital Accounting Research* (2018), pp. 161–188 (cit. on p. 64).

[Rae+19] Jack W Rae et al. « Compressive transformers for long-range sequence modelling ». In: *arXiv preprint arXiv:1911.05507* (2019) (cit. on p. 46).

[Raf+20] Colin Raffel et al. « Exploring the limits of transfer learning with a unified text-to-text transformer ». In: *Journal of machine learning research* 21.140 (2020), pp. 1–67 (cit. on p. 46).

[Ran+18] Syama Sundar Rangapuram et al. « Deep State Space Models for Time Series Forecasting ». In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018 (cit. on p. 64).

[Ros58] F. Rosenblatt. « The perceptron: A probabilistic model for information storage and organization in the brain. » In: *Psychological Review* 65 (1958), pp. 386–408. ISSN: 0033-295X (cit. on pp. 31, 73, 152).

[Roy+21] Aurko Roy et al. « Efficient content-based sparse attention with routing transformers ». In: *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 53–68 (cit. on p. 46).

[Rug76] Alan M Rugman. « Risk reduction by international diversification ». In: *Journal of International Business Studies* 7 (1976), pp. 75–80 (cit. on p. 127).

[SY14] Mayu Sakurada and Takehisa Yairi. « Anomaly detection using autoencoders with nonlinear dimensionality reduction ». In: *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*. 2014, pp. 4–11 (cit. on pp. 23, 184).

[SI22] Hugo Schnoering and Hugo Inzirillo. « Deep Fusion of Lead-lag Graphs: Application to Cryptocurrencies ». In: *arXiv preprint arXiv:2201.02040* (2022) (cit. on p. 126).

[SGO20] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. « Financial time series forecasting with deep learning: A systematic literature review: 2005–2019 ». In: *Applied soft computing* 90 (2020), p. 106181 (cit. on p. 30).

[SSL19] Shun-Yao Shih, Fan-Keng Sun, and Hung-yi Lee. « Temporal pattern attention for multivariate time series forecasting ». In: *Machine Learning* 108 (2019), pp. 1421–1441 (cit. on p. 45).

[Sho+19] Mohammad Shoeybi et al. « Megatron-lm: Training multi-billion parameter language models using model parallelism ». In: *arXiv preprint arXiv:1909.08053* (2019) (cit. on p. 46).

[STN18] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. « A comparison of ARIMA and LSTM in forecasting time series ». In: *2018 17th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2018, pp. 1394–1401 (cit. on p. 176).

[SLL19] David So, Quoc Le, and Chen Liang. « The evolved transformer ». In: *International conference on machine learning*. PMLR. 2019, pp. 5877–5886 (cit. on p. 46).

[SCS19] Jung Yoon Song, Woojin Chang, and Jae Wook Song. « Cluster analysis on the structure of the cryptocurrency market via Bitcoin–Ethereum filtering ». In: *Physica A: Statistical Mechanics and its Applications* 527 (2019), p. 121339 (cit. on p. 126).

[Sor09] Didier Sornette. *Why stock markets crash: critical events in complex financial systems*. Princeton university press, 2009 (cit. on p. 22).

[SM19] Ralf C Staudemeyer and Eric Rothstein Morris. « Understanding LSTM–a tutorial into long short-term memory recurrent neural networks ». In: *arXiv preprint arXiv:1909.09586* (2019) (cit. on pp. 35, 47).

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. « Sequence to sequence learning with neural networks ». In: *Advances in neural information processing systems* 27 (2014).

[TSB10] Souhaib Ben Taieb, Antti Sorjamaa, and Gianluca Bontempi. « Multiple-output modeling for multi-step-ahead time series forecasting ». In: *Neurocomputing* 73.10-12 (2010), pp. 1950–1957 (cit. on pp. 5, 44).

[TDF91] Zaiyong Tang, Chrys De Almeida, and Paul A Fishwick. « Time series forecasting using neural networks vs. Box-Jenkins methodology ». In: *Simulation* 57.5 (1991), pp. 303–310 (cit. on pp. 5, 44, 151).

[Tay+20] Yi Tay et al. « Sparse sinkhorn attention ». In: *International Conference on Machine Learning*. PMLR. 2020, pp. 9438–9447.

[TT89] George C Tiao and Ruey S Tsay. « Model specification in multivariate time series ». In: *Journal of the Royal Statistical Society: Series B (Methodological)* 51.2 (1989), pp. 157–195 (cit. on pp. 5, 44).

[Vac+24] Cristian J Vaca-Rubio et al. « Kolmogorov-arnold networks (kans) for time series analysis ». In: *arXiv preprint arXiv:2405.08790* (2024) (cit. on pp. 45, 151).

[Vas+17] Ashish Vaswani et al. *Attention Is All You Need*. 2017 (cit. on pp. 1, 5, 44, 46, 51, 151, 170, 174, 184, 188, 189).

[Wan+19a] Renzhuo Wan et al. « Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting ». In: *Electronics* 8.8 (2019), p. 876 (cit. on p. 45).

[Wan+19b] Kang Wang et al. « Multiple convolutional neural networks for multivariate time series prediction ». In: *Neurocomputing* 360 (2019), pp. 107–119.

[Wan+20] Sinong Wang et al. « Linformer: Self-attention with linear complexity ». In: *arXiv preprint arXiv:2006.04768* (2020).

[Wan+16] Yequan Wang et al. « Attention-based LSTM for aspect-level sentiment classification ». In: *Proceedings of the 2016 conference on empirical methods in natural language processing*. 2016, pp. 606–615 (cit. on p. 170).

[Wei18] William WS Wei. *Multivariate time series analysis and applications*. John Wiley & Sons, 2018 (cit. on pp. 5, 44, 151).

[Wer90] Paul J Werbos. « Backpropagation through time: what it does and how to do it ». In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560 (cit. on p. 30).

[Wil89] Ronald J Williams. « A learning algorithm for continually running fully recurrent neural netwokrs ». In: *Neural Computation* 1 (1989), pp. 256–263 (cit. on p. 30).

[Wu+20a] Neo Wu et al. « Deep transformer models for time series forecasting: The influenza prevalence case ». In: *arXiv preprint arXiv:2001.08317* (2020) (cit. on p. 46).

[Wu+20b] Sifan Wu et al. « Adversarial sparse transformer for time series forecasting ». In: *Advances in neural information processing systems* 33 (2020), pp. 17105–17115 (cit. on p. 46).

[Wu+16] Yonghui Wu et al. « Google's neural machine translation system: Bridging the gap between human and machine translation ». In: *arXiv preprint arXiv:1609.08144* (2016).

[Wu+20c] Zonghan Wu et al. « Connecting the dots: Multivariate time series forecasting with graph neural networks ». In: *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2020, pp. 753–763.

[XC21] Xiuqin Xu and Ying Chen. « Deep Switching State Space Model (DSSSM) for Nonlinear Time Series Forecasting with Regime Switching ». In: *arXiv preprint arXiv:2106.02329* (2021).

[Yan+21] Bin Yan et al. « Learning spatio-temporal transformer for visual tracking ». In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10448–10457.

[Yan+19] Hang Yan et al. « TENER: adapting transformer encoder for named entity recognition ». In: *arXiv preprint arXiv:1911.04474* (2019) (cit. on p. 46).

[Yan+22] Weixin Yang et al. « Developing the path signature methodology and its application to landmark-based human action recognition ». In: *Stochastic Analysis, Filtering, and Stochastic Optimization: A Commemorative Volume to Honor Mark HA Davis's Contributions*. Springer, 2022, pp. 431–464 (cit. on pp. 15, 128, 151).

[Yan+16] Zichao Yang et al. « Hierarchical attention networks for document classification ». In: *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2016, pp. 1480–1489.

[Ye+19] Zihao Ye et al. « Bp-transformer: Modelling long-range context via binary partitioning ». In: *arXiv preprint arXiv:1911.04070* (2019).

[Yu+20] Jianfei Yu et al. « Improving multimodal named entity recognition via entity span detection with unified multimodal transformer ». In: Association for Computational Linguistics. 2020 (cit. on p. 46).

[Zen+22] Ailing Zeng et al. *Are Transformers Effective for Time Series Forecasting?* 2022 (cit. on p. 170).

[Zen+23] Ailing Zeng et al. « Are transformers effective for time series forecasting? » In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 37. 9. 2023, pp. 11121–11128.

[Zey+19] Albert Zeyer et al. « A Comparison of Transformer and LSTM Encoder Decoder Models for ASR ». In: 2019, pp. 8–15. DOI: 10.1109/ASRU46091.2019.9004025 (cit. on p. 184).

[Zha+21a] Shuangfei Zhai et al. « An Attention Free Transformer ». In: *CoRR* abs/2105.14103 (2021) (cit. on pp. 25, 170, 172, 174, 175, 184, 188, 189).

[Zha+21b] Hang Zhang et al. « Poolingformer: Long document modeling with pooling attention ». In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12437–12446.

[Zha+19] Xuan Zhang et al. « At-lstm: An attention-based lstm model for financial time series prediction ». In: *IOP Conference Series: Materials Science and Engineering*. Vol. 569. 5. IOP Publishing. 2019, p. 052037 (cit. on p. 170).

[Zha+17] Yuhao Zhang et al. « Position-aware attention and supervised data improve slot filling ». In: *Conference on empirical methods in natural language processing*. 2017.

[ZP17] Chong Zhou and Randy C Paffenroth. « Anomaly detection with robust deep autoencoders ». In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 665–674 (cit. on pp. 23, 184).

[Zho+21] Haoyi Zhou et al. « Informer: Beyond efficient transformer for long sequence time-series forecasting ». In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 12. 2021, pp. 11106–11115 (cit. on p. 46).

[ZW06] Eric Zivot and Jiahui Wang. « Vector autoregressive models for multivariate time series ». In: *Modeling financial time series with S-PLUS®* (2006), pp. 385–429 (cit. on pp. 5, 44, 151).

ECOLE
DOCTORALE
DE MATHEMATIQUES
HADAMARD

Titre : Contributions aux Méthodes Econométriques et d'Apprentissage Profond pour la Prévision de Séries Temporelles

Mots clés : Apprentissage profond, économétrie non linéaire, apprentissage machine, intelligence artificielle, path signatures, séries temporelles

Résumé : Cette thèse, structurée en quatre parties distinctes, contribue à enrichir les domaines de l'apprentissage profond et de l'économétrie non linéaire. Les modèles traditionnels ont souvent montré des faiblesses pour capturer les comportements non linéaires et les changements de régime caractéristiques des actifs numériques. Cette thèse se concentre sur l'amélioration de la prédiction des rendements et de la volatilité de cette nouvelle classe d'actifs à travers le développement de nouvelles architectures de réseaux de neurones et de méthodologies innovantes. Dans une première partie, nous introduisons les réseaux de Kolmogorov-Arnold temporels (TKAN) ainsi qu'une nouvelle architecture de type transformer (TKAT) reposant sur les TKAN pour la prévision des séries temporelles. Ces modèles intègrent la gestion de la mémoire et le mécanisme d'attention pour capturer les dynamiques complexes des marchés financiers. Nos résultats montrent que le TKAN et le TKAT surpassent les modèles récurrents traditionnels (LSTM, GRU) en termes de stabilité et de précision de prédiction, particulièrement sur des horizons de temps plus longs. La deuxième partie propose une nouvelle approche de modélisation des régimes de marché à l'aide de réseaux de neurones récurrents et des nouveaux modèles GARCH à changements de régimes Markoviens. La première approche permet de détecter les régimes de marché et d'estimer les probabilités de transition entre les différents états. Les résultats montrent une amélioration significative dans la détection et la prédiction des régimes de marché par rapport aux modèles traditionnels. Dans la seconde famille de modèles, de nouvelles spécifications de type MS-GARCH sont définies; des procédures d'inférence statistique par simulation sont proposées et testées. Dans la troisième partie, nous explorons l'utilisation des signatures pour la construction de portefeuilles. Une méthode de classification basée sur les signatures est développée pour sélectionner les actifs les plus représentatifs de chaque cluster, améliorant ainsi le ratio rendement-risque. Les portefeuilles construits avec cette méthodologie montrent des performances supérieures par rapport aux portefeuilles standards. La dernière partie de cette thèse aborde la modélisation de la volatilité et la détection des anomalies. Nous proposons une nouvelle couche LSTM, l'Attention Free Long Short-Term Memory (AF-LSTM), pour la prédiction de la volatilité. Cette nouvelle couche montre des performances prédictives supérieures à la version originale. Pour la détection des anomalies, un auto-encodeur conditionnel (AF-CAE) est introduit, offrant une meilleure détection des anomalies, réduisant les faux positifs. Cette thèse propose plusieurs contributions dans le domaine de la prédiction des séries temporelles les résultats démontrent des améliorations en termes de précision et de robustesse, ouvrant de nouvelles perspectives pour la modélisation et la gestion des actifs financiers.

Title : Contributions to Econometric and Deep Learning Methods for Time Series Forecasting

Keywords : Deep Learning, nonlinear econometrics, machine learning, artifical intelligence, path signatures, timeseries

Abstract : This thesis, structured in four distinct parts, contributes to enriching the fields of deep learning and nonlinear econometrics. Traditional models have often shown weaknesses in capturing the nonlinear behaviors and regime shifts characteristic of digital assets. This thesis focuses on improving the prediction of returns and volatility of this new asset class through the development of new neural network architectures and innovative methodologies. In a first part, we introduce Temporal Kolmogorov-Arnold Networks (TKANs) as well as a new transformer-based architecture (TKAT) based on TKANs for time series forecasting. These models integrate memory management and attention mechanism to capture the complex dynamics of financial markets. Our results show that TKAN and TKAT outperform traditional recurrent models (LSTM, GRU) in terms of stability and prediction accuracy, particularly over longer time horizons. The second part proposes a new approach for modeling market regimes using recurrent neural networks and new Markov-Switching GARCH dynamics. This former approach allows to detect market regimes and estimate transition probabilities between different states. The results show a significant improvement in the detection and prediction of market regimes compared to traditional models. The latter models define new meaningful MS-GARCH specifications and convenient simulation-based estimations procedures. In the third part, we explore the use of signatures for portfolio construction. A signature-based classification method is developed to select the most representative assets of each cluster, thus improving the risk-return ratio. Portfolios built with this methodology show superior performances compared to standard portfolios. The last part of this thesis addresses volatility modeling and anomaly detection. We propose a new LSTM layer, the Attention Free Long Short-Term Memory (AF-LSTM), for volatility prediction. This new layer shows superior predictive performances compared to the original version. For anomaly detection, a conditional autoencoder (AF-CAE) is introduced, providing better anomaly detection, reducing false positives. This thesis proposes several contributions in the field of time series prediction; the results demonstrate improvements in terms of accuracy and robustness, opening new perspectives for the modeling and management of financial assets.

Institut Polytechnique de Paris
91120 Palaiseau, France