Creating Investment Strategies Based on Machine Learning Algorithms

Hachem Madmoun

▶ **To cite this version:**

Hachem Madmoun. Creating Investment Strategies Based on Machine Learning Algorithms. Probability [math.PR]. École des Ponts ParisTech, 2022. English. NNT : 2022ENPC0037 . tel-04861182

HAL Id: tel-04861182
https://pastel.hal.science/tel-04861182v1

Submitted on 2 Jan 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

École des Ponts ParisTech

THÈSE DE DOCTORAT

de l'École des Ponts ParisTech

Creating Investment Strategies based on Machine Learning Algorithms

École doctorale des Ponts et Chaussées

Mathématiques appliquées

Thèse préparée au Cermics

Thèse soutenue le 30 Novembre 2022, par

Hachem MADMOUN

Composition du jury:

Agnès, SULEM Directrice de Recherche, Inria	*Président*
Nabil, KAHALE Professeur, ESCP Business School	*Rapporteur*
Tristan, CAZENAVE Professeur, Université de Dauphine	*Rapporteur*
François, LONGIN Professeur, ESSEC Business School	*Examinateur*
Romuald, ELIE Professeur, DeepMind & UGE	*Invité*
Bernard, LAPEYRE Professeur, Ecole des Ponts et Chaussées	*Directeur de thèse*

Remerciements

Je remercie toute l'équipe de chercheurs du Cermics pour qui j'ai la plus grande estime, ainsi que M. Romuald Elie dont le cours m'a donné envie de faire du Machine Learning pour le reste de ma vie. Son talent et sa pédagogie sont une immense source d'inspiration. Un grand merci également à Isabelle pour son incroyable empathie et à M. Kahalé pour sa relecture minutieuse et ses remarques pertinentes.

Je tiens également à adresser mes plus chaleureux remerciements à mon directeur de thèse. Il n'existe pas de mots assez forts pour décrire toute l'admiration qu'il m'inspire. Bien que je lui sois infiniment reconnaissant pour tout le savoir mathématique qu'il m'a si généreusement transmis, ce n'est pas ce que je retiendrai de plus précieux après ces nombreuses années passées à ses côtés. Je retiendrai davantage les discussions, les cafés, sa passion contagieuse pour la recherche, son attachement à faire en sorte que les étudiants trouvent leurs voies. Ce n'est pas qu'un merveilleux intercesseur de connaissances, c'est surtout un allié de taille dont les qualités humaines sont sans commune mesure. Est bien heureux quiconque a la chance de croiser son chemin durant sa scolarité.

Un grand merci également à l'ensemble des professeurs de mathématiques qui ont illuminé mon enfance : M. Durand, M. Gretet, M. Kharbat, M. Dfaili et enfin ma mère qui fut le tout premier professeur à m'insuffler une passion pour les mathématiques.

J'ai une pensée toute particulière pour mon frère, ce grand amoureux d'informatique qui m'abreuvait de concepts ésotériques avant même que je pronnonce mes premiers mots. Je remercie enfin mon incroyable soeur que j'aime si fort, mes collègues et très bons amis Timothée et Jeremy avec qui j'ai passé d'excellents moments, ainsi que Najat, Sarah, Salem et Rachid, ma famille de coeur.

Résumé de la thèse en anglais

The goal of this thesis is to show how portfolio allocation can benefit from the development of Machine Learning algorithms. We propose two investment strategies based on such algorithms.

The first strategy (Low Turbulence Model) is an asset allocation method based on an interpretable low-dimensional representation framework of financial time series combining signal processing, deep neural networks, and bayesian statistics. This representation framework exhibits particularly good clustering properties, which enables us to define two market regimes : The High and the Low Turbulence regimes. By modeling their dynamics, we are able to forecast the market regime over the next period of time. Hence, we enrich the paradigm of asset management by reducing investable periods for any risky asset to their low turbulence periods and alternatively invest in safe assets (typically cash or treasuries) during high turbulence periods. By doing so, we not only enable investors to enjoy smoother "investment journeys", but we also aim at generating reasonable returns by considerably reducing the volatility and the drawdowns along the way.

The Second Strategy (Attention Based Ranking Model) is a stock picking strategy for large-cap US stocks. Our approach involves two steps : First, we transform the series of price data into skill vectors using the TrueSkill Algorithm with Optimal Quantized Belief Propagation. An Attention-based Recurrent Neural Network is then applied to selectively focus on relevant parts of the skill vectors in order to predict the ranking of a group of assets over the next period of time. By doing so, the ranking system learns to adapt to different market configurations and significantly outperforms both the market and classic momentum strategies.

Keywords— Portfolio Management, Time Frequency Analysis, Sequential Models, Generative Models, Bayesian Learning, Attention Mechanisms

Table des matières

Table des figures

Liste des tableaux

1 | Introduction

1.1 Contexte de la thèse

Harry Markowitz a développé son modèle de construction de portefeuille en 1952. Pourtant, quasiment soixante-dix ans après, on y fait toujours référence comme étant la théorie *moderne* de portefeuille. S'il est vrai que l'on parlait déjà de risque et de rendement avant Markowitz, c'est bien à lui que l'on doit une définition rigoureuse de ces concepts et l'introduction d'un formalisme en matière de construction de portefeuille. On pourrait ainsi dire que Markowitz est au monde de la finance ce que Descartes est à la philosophie moderne, à savoir celui qui a eu le mérite d'introduire le jargon, d'amorcer une discipline balbutiante, tout en proposant des réponses au succès mitigé. De fait, la théorie du portefeuille de Markowitz n'est que très peu utilisée dans l'industrie.

Les algorithmes de Machine Learning ont quant à eux généré un enthousiasme certain au sein de la communauté scientifique dans des domaines aussi variés que le langage, la parole, la vision ou même la biologie. L'objectif de cette thèse est d'explorer un certain nombre d'idées issues du Machine Learning, d'en saisir les mécanismes afin de les adapter à un domaine d'application récent : la gestion d'actifs. En d'autres termes, il s'agit de conjuguer harmonieusement différentes intuitions afin de proposer un nouveau paradigme d'investissement.

1.2 Contributions

Durant la thèse, fruit d'une collaboration entre le laboratoire Bramham Gardens et le Cermics, nous avons développé deux approches qui prennent le contrepied des méthodes d'investissement classiques. En voici un bref descriptif.

— **Low Turbulence Model** :

Il s'agit d'une stratégie sur indice qui combine à la fois du traitement du signal et des modèles graphiques afin de créer une représentation en basse dimension des séries temporelles. Cette représentation permet d'identifier deux régimes de marché, stables dans le temps, associés à l'indice en question : un régime de forte turbulence et un régime de faible turbulence. L'étude de la dynamique de ces régimes nous permet de prédire l'état de turbulence futur d'un indice et d'ainsi réduire les périodes d'investissement aux périodes de faible turbulence qui lui sont associées. Un tel paradigme d'investissement permet d'éviter au maximum les périodes de crise, sans pour autant sacrifier le niveau de rendement.

— **Attention Based Ranking Model** :

Il s'agit d'une stratégie de sélection de stocks, développée conjointement avec Jeremy Chichportich et son directeur de thèse Idriss Kharroubi. Le modèle prédit le classement futur des actifs en s'appuyant sur la dynamique de leurs skills, estimés grâce à un réseau bayésien. L'étude de la dynamique des skills repose, quant à elle, sur l'utilisation de mécanismes d'attention qui permettent aux réseaux de neurones de se focaliser sur la partie la plus pertinente de l'information. On retrouve d'ailleurs cette logique de pondération asymétrique de l'information en neuroscience et en psychologie pour qualifier le processus de raisonnement lent, réfléchi et logique, auquel l'humain a recours pour résoudre des problèmes complexes. Ainsi, en s'inspirant de ces architectures récentes, qui ont fait le succès des modèles de langage naturel, nous avons proposé un modèle à même de s'adapter aux différentes configurations de marché en brisant la propension au pur momentum. De fait, le modèle surperforme significativement le marché ainsi que les stratégies momentum classiques.

1.3 Organisation de la thèse

Le document de thèse se compose de trois chapitres. Le chapitre 2 introduit les bases des méthodes séquentielles qui sous-tendent les modèles développés, illustrées dans le domaine du langage. Les chapitres 3 et 4 introduisent les stratégies d'investissement **Low Turbulence** et **Attention Based Ranking Model**.

1.3.1 Chapitre 2 : Introducing Sequential Models with Applications in Natural Language Processing

Les modèles de Markov cachés (HMM) jouissaient d'une forte popularité dans les années 1980 de par leur structure mathématique séduisante et leurs bonnes performances dans un large éventail d'applications. Cependant, ils reposent sur une hypothèse de Markov quant à la structure temporelle de leurs états cachés, ce qui rend le modèle inefficient pour modéliser les dépendances de long terme.

L'utilisation des HMMs a peu à peu été éclipsée par l'émergence d'une nouvelle classe de méthodes : les réseaux de neurones récurrents (RNNs), utilisés par exemple dans la reconnaissance vocale (Rabiner & Juang, 1986). Si les RNNs proposent une gestion de la mémoire plus riche que les HMMs, ces modèles souffrent cependant de problèmes de gradient évanescent (Pascanu, Mikolov, & Bengio, 2013). De nombreuses architectures ont donc été proposées pour pallier ces difficultés et améliorer la prise en compte de dépendances de long terme. La première architecture de ce type, appelée Long Short-Term Memory (LSTM), a été proposée par Hochreiter et Schmidhuber en 1997 (Hochreiter & Schmidhuber, 1997) puis affinée dans (Gers, Schmidhuber, & Cummins, 2000). Plus récemment, le modèle Gated Recurrent Unit

(GRU) (Cho et al., 2014) a été introduit dans le contexte de la traduction automatique.

Les modèles LSTM et GRU ont servi de brique de base à de nombreuses architectures plus complexes visant à associer une séquence d'entrée à une séquence de sortie. S'est alors posée la question de l'alignment optimal entre les deux séquences. Cette problématique a donné naissance aux mécanismes d'attention.

Ainsi, le modèle d'attention, initialement introduit pour la traduction automatique (Bahdanau, Cho, & Bengio, 2017), est devenu extrêmement populaire au sein de la communauté de l'intelligence artificielle jusqu'à devenir l'élément incontournable des réseaux de neurones dans pratiquement tous les domaines. Qu'il s'agisse du traitement du langage naturel (Natural Language Processing) (Sood, Tannert, Müller, & Bulling, 2020), de la vision (Gou, Yu, Maybank, & Tao, 2021) ou du traitement de la parole (Speech Recognition) (Cho, Courville, & Bengio, 2015), il y a une forme de concensus sur l'efficacité des modèles d'attention, issus des neurosciences.

Les pionniers de l'intelligence artificielle y voient même la pierre angulaire d'un basculement d'une intelligence artificielle qui prédit à partir de l'identification de patterns dans les données à une intelligence artificielle qui calque le raisonnement humain, notamment du système 2 défini par Daniel Kahneman dans (Daniel, 2017).

L'intuition qui sous-tend l'attention repose sur les systèmes biologiques humains. Notre système de traitement visuel a tendance à se focaliser sur certaines parties de l'image, tout en ignorant d'autres informations non pertinentes à la perception (Xu et al., 2015). Cette prise en compte sélective de l'information pertinente se révèle cruciale dans bon nombre de problèmes tels que le langage, la parole ou la vision. A titre d'exemple, dans les tâches consistant à produire une traduction automatique ou le résumé d'un ouvrage, seuls certains mots de la séquence d'entrée doivent être pris en compte pour prédire le mot suivant. La figure 1.1, citée dans (Bahdanau et al., 2017) illustre ce phénomène de pondération asymétrique de l'information d'entrée dans le cadre de la traduction.

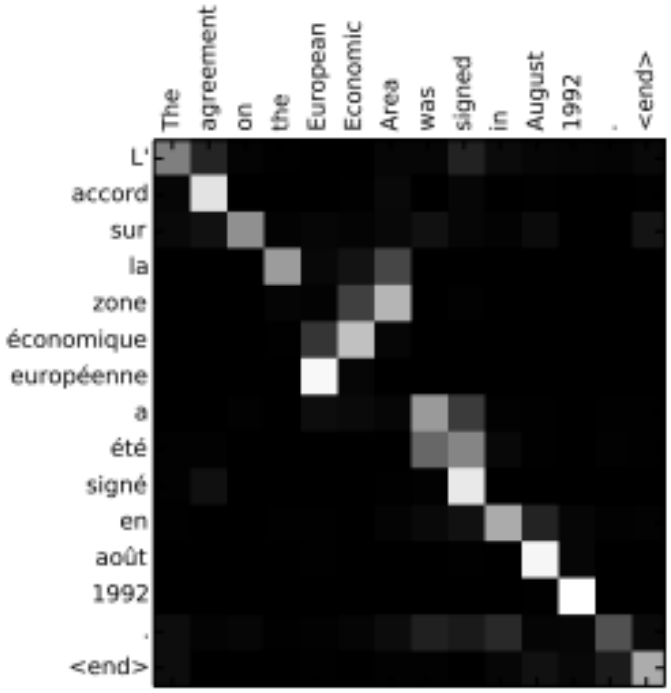


FIGURE 1.1 – Prise en compte partielle de l'information par l'attention en traduction

L'apprentissage d'un alignement optimal entre l'information fournie en entrée et l'information prédite représente un enjeu majeur en traduction puisqu'elle permet de mieux tenir compte de la diversité des structures grammaticales d'une langue à une autre et d'une meilleure contextualisation de l'information. Ce concept d'alignement repose sur le calcul de poids α qui permettent d'introduire une asymétrie dans la prise en compte de l'information donnée en entrée.

Pour schématiser le concept d'alignmenent, la figure 1.2 en présente les principales étapes. Les données d'entrée sont représentées par les vecteurs $k_1, \ldots, k_n$, le décodeur vient questionner les données d'entrée à l'aide d'un vecteur q (query) en calculant une mesure de similarité entre la query et les vecteurs (k_i). La normalisation de ces similarités permet de définir les poids de l'attention.

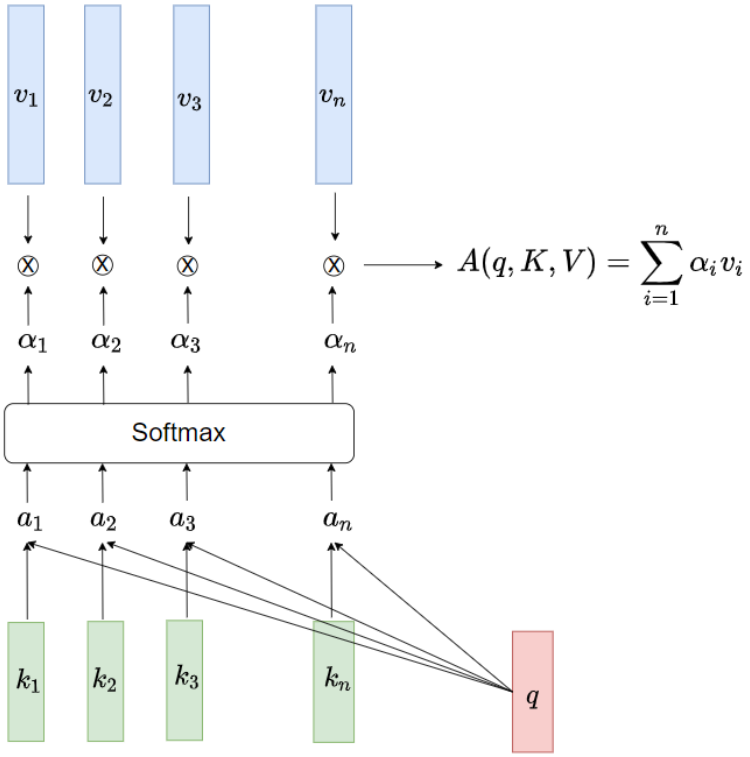


FIGURE 1.2 – Illustration des poids de l'attention

Ainsi, de nombreux papiers, tels que (Britz, Goldie, Luong, & Le, 2017), (Tang, Müller, Rios, & Sennrich, 2018) et (B. Zhang, Xiong, Xie, & Su, 2020) ont mis en évidence l'apport de l'attention dans le domaine de la traduction, notamment lorsqu'il s'agit de traiter de longues séquences.

Il était devenu d'usage d'utiliser les modèles séquentiels tels que le LSTM ou le GRU dans l'encodeur et le décodeur, puis un module d'attention s'interposait entre les deux pour gérer l'alignement optimal entre les sorties de l'encodeur et les entrées du décodeur.

Une étape majeure dans le domaine de la réprésenation séquentielle a été franchie lorsqu'en 2017, Google a présenté le modèle "Transformer" dans le papier (Vaswani et al., 2017), sobrement intitulé "*Attention is all you need*", suggérant de laisser tomber les LSTMs, pour ne plus avoir qu'une représentation en trois blocs d'attention, un pour l'encodeur, un pour le décodeur et un pour la gestion de l'alignement optimal. Les 3 blocs d'attention sont représentés en orange dans la figure 1.3

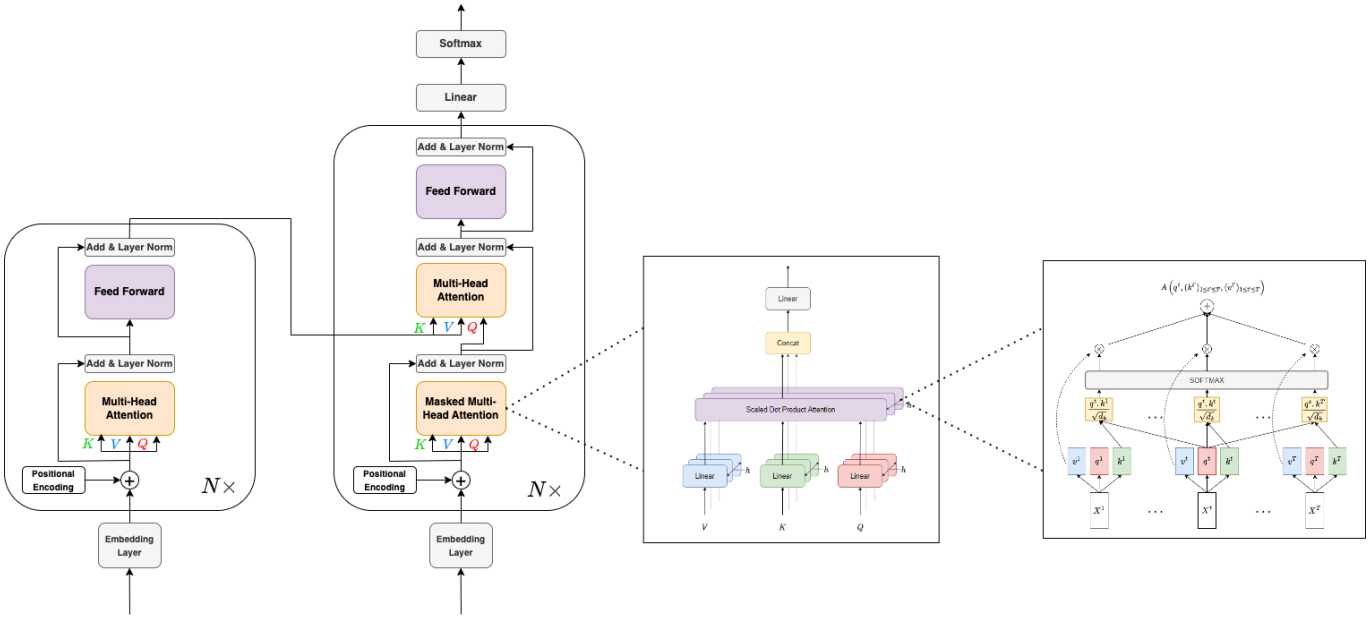


FIGURE 1.3 – Le transformer

Depuis, de nombreux modèles de langage ont été proposés en se basant sur l'architecture du Transformer parmi lesquels **GPT-2** (Chen et al., 2020) et **GPT-3** (Brown et al., 2020), introduits par Open AI, le modèle **BERT** (Kenton & Toutanova, 2019) proposé par Google AI Language et le modèle **T5** (Raffel et al., 2020) proposé également par Google. Ces modèles de langage sont entraînés sur des quantités astronomiques de données pour ensuite être utilisés dans diverses applications.

Ainsi, le modèle **BERT**, représenté dans la figure 1.4, illustre la façon dont on peut mettre à profit un même modèle de langage pour réaliser plusieurs tâches : de la classification entre paires de séquences (a), de la classification associée à une seule séquence (b), l'entraînement à répondre à des questions (Question answering) (c), ainsi que de la prédiction de tags associés à une séquence (sentence tagging) (d).

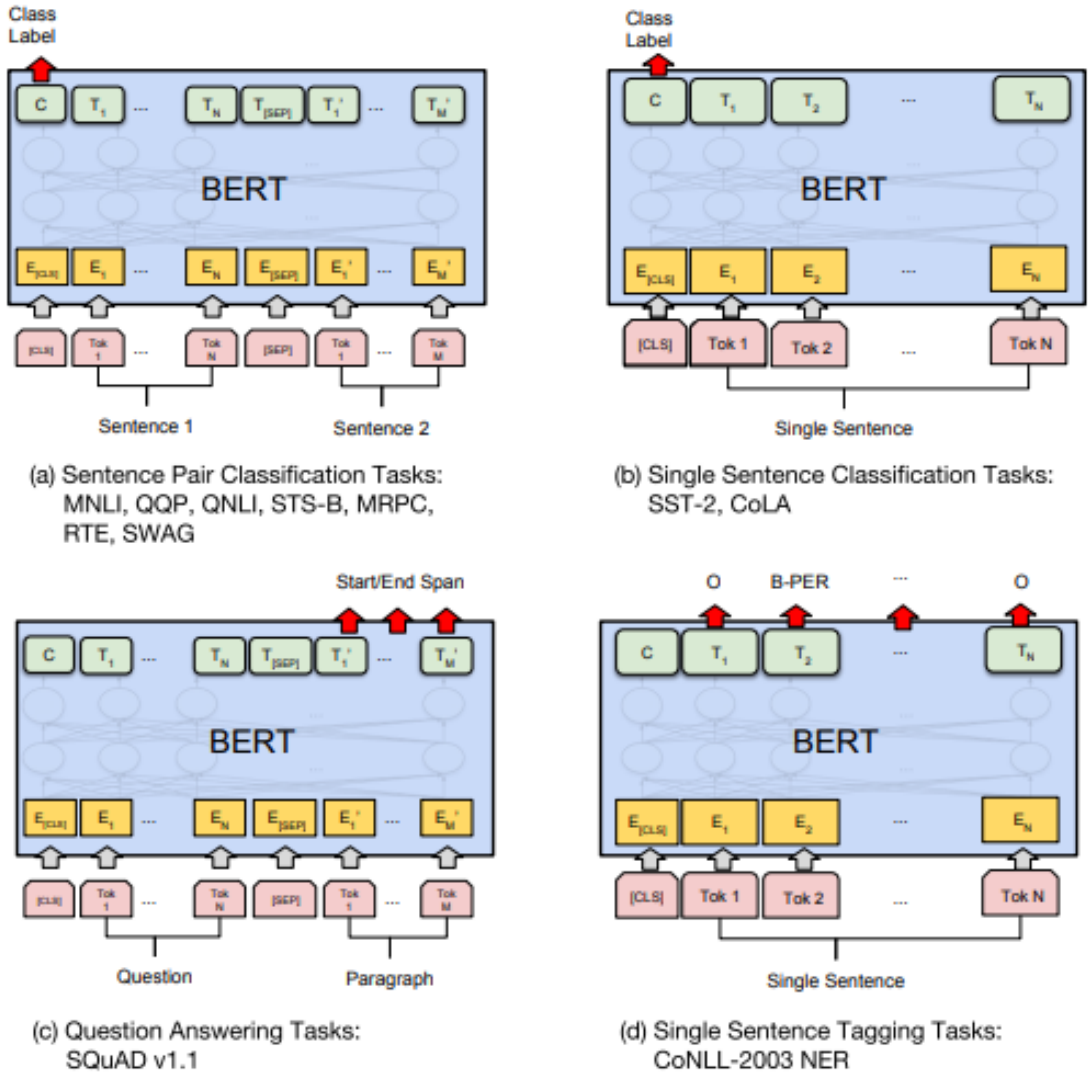


FIGURE 1.4 – Le modèle BERT utilisé pour différentes tâches en NLP

De façon analogue à ce qui se passe dans le langage, lorsqu'il s'agit de produire la légende d'une image (Image Captioning), certaines régions de l'image d'entrée peuvent être plus pertinentes pour générer le prochain mot de la légende. Les modèles d'attention intègrent cette notion de pertinence en permettant au modèle de prêter dynamiquement attention à certaines parties de l'entrée qui aident à réaliser efficacement la tâche à accomplir, comme le suggère la figure 1.5, dérivée de (Xu et al., 2015).

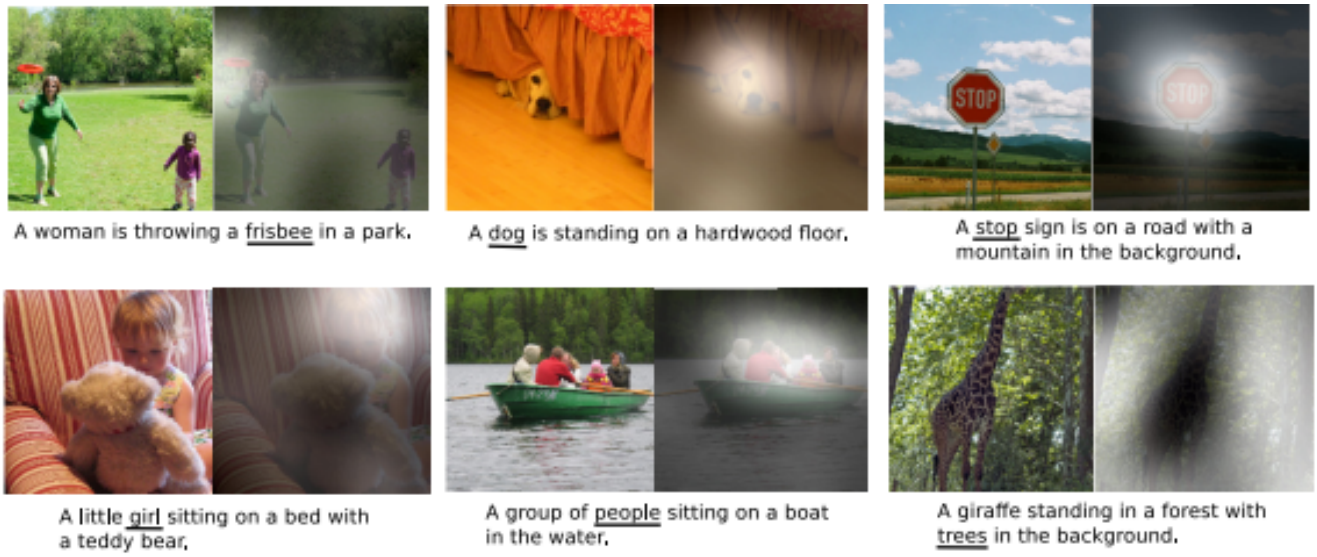


FIGURE 1.5 – Prise en compte partielle de l'information par l'attention en vision

La figure 1.6, issue du blog de Chris Olah, illustre l'utilisation de l'attention dans le domaine de la parole. On y observe en entrée les vecteurs acoustiques représentant la parole sur différentes fenêtres d'environ 20 ms (ce qui consiste à grands traits à appliquer une transformation de Fourier fenêtrée au signal puis d'en agréger le résultats en bandes de fréquences). Traditionnellement, la reconnaissance de la parole se faisait à l'aide des méthodes de Makov cachés dont les observations sont les vecteurs acoustiques et dont les variables latentes sont les phonèmes. Le souci avec une telle modélisation est qu'elle impose un alignement parfait entre le vecteur acoustique et le phonème associé. L'utilisation de l'attention a permis d'avoir plus de flexibilité puisque désormais, un phonème donné n'est plus contraint de se focaliser entièrement sur le vecteur acoustique associé, mais peut également pondérer les vecteurs acoustiques adjacents.

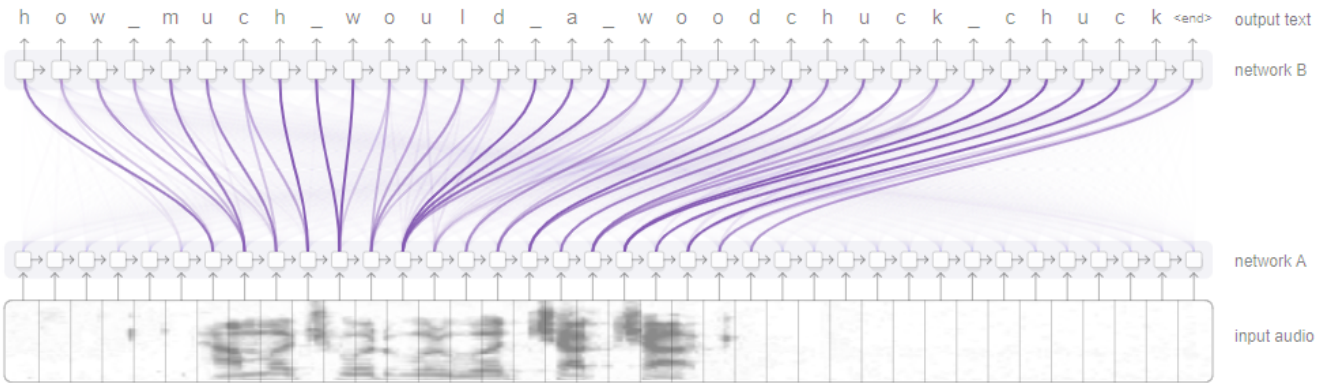


FIGURE 1.6 – Prise en compte partielle de l'information par l'attention dans le domaine de la parole

Enfin, le domain de la biologie a également été révolutionné par les modèles d'attention (Senior et al.,

2020). Ainsi, en remplaçant les modèles de convolution (**Alphafold1** en 2018) par les modèles d'attention (**Alphafold2** en 2020), DeepMind a amélioré son modèle en atteignant la meilleure performance dans la compétition CASP sur la problématique du repliement des proétines, un challenge qui date de plus de 50 ans dans le domaine de la biologie.

Le chapitre présente enfin la problématique d'optimisation des poids de ces réseaux de neurones grâce à l'algorithme Adam. L'algorithme a été introduit pour la première fois en 2015 (Kingma & Ba, 2015). Les auteurs ont proposé une preuve de convergence qui s'est avérée problématique. En 2018, (S. J. Reddi, Kale, & Kumar, 2018) a clarifié l'incohérence de l'article précédent et a proposé une preuve dans le cadre convexe. (S. Reddi, Zaheer, Sachan, Kale, & Kumar, 2018) a quant à lui traité le cas non convexe. Nous avons restructuré la preuve en plusieurs étapes pour la rendre plus lisible au lecteur.

1.3.2 Chapitre 3 : Forecasting Market turbulence regimes using Latent Spectral representation

S'il est relativement aisé de capturer la signature d'une symphonie de Beethoven en analysant la décomposation du morceau en différentes harmoniques, on peut imaginer que la structure harmonique d'un signal financier puisse contenir de l'information exploitable. La complexité du marché se retrouverait ainsi décrite par l'amplitude des différentes harmoniques, ainsi que par leur structure d'interdépendance.

Afin d'analyser la structure harmonique du signal financier, nous nous sommes inspirés des modèles génératifs, souvent utilisés dans le domaine de la vision. L'idée sous-jacente consiste à apprendre à compresser une image pour en extraire la substantifique moelle avant de la recomposer à l'identique à partir de sa version compressée. Prenons l'exemple d'une image représentant un visage. La structure d'interdépendance entre les différents pixels est éminemment complexe. Le modèle aura pour but de décrire cette structure en un nombre extrêmement réduit de caractéristiques, chacune représentant un trait du visage. Ce qui permettrait de générer la reconstruction précise du visage à partir de ces caractéristiques comme on dresserait un portrait robot à partir de divers témoignages. Nous nous sommes donc inspirés de ce type de modèle pour décrire la structure des harmoniques en un nombre extrêmement réduit de caractéristiques.

En purifiant l'information de nos signaux à travers les transformations sus-énoncées, nous parvenons à en extraire une représentation condensée et exploitable. Une telle représentation nous permet notamment d'identifier deux régimes de marché parfaitement séparés (en ratio de Sharpe notamment) et stables dans le temps, de sorte que l'on passe d'un signal financier extrêmement complexe à une séquence binaire riche en information et dont la stabilité confère sa prédictibilité. Tout se passe comme si l'on avait forcé le marché à s'exprimer en un vocabulaire réduit aux deux syntagmes "forte turbulence" et "basse

turbulence", le but étant de prédire le prochain syntagme.

Les états de turbulence étant intrinsèques à chaque signal financier, la démarche peut être déployée à un spectre extrêmement large d'applications. Si l'on prend l'exemple d'un indice financier dans une géographie particulière, il devient tout à fait aisé de proposer une stratégie d'investissement dont l'exposition à l'indice dépend de la prédiction du régime de turbulence. Les résultats obtenus mettent en évidence une forte protection contre les drawdowns, sans pour autant sacrifier le niveau de rendement.

1.3.3 Chapitre 4 : Bayesian Approach with Attention Mechanism for Learning to Rank Financial Assets

Les avancées majeures en NLP découlent principalement de l'intuition du linguiste John Rupert Firth : *"You shall know a word by the company it keeps"*. Ainsi, de nombreux algorithmes ont traduit cette intuition par différents formalismes afin de créer une représentation multidimensionnelle d'un mot que l'on nomme **embedding**, riche en information sémantique, définie à partir des voisins de ce mot. Comme souvent en apprentissage non supervisé, il est difficile d'interpréter les différentes dimensions du vecteur embedding, mais l'on arrive tout de même à vérifier que certaines propriétés d'analogie y sont encodées. Typiquement, l'analogie "king is to queen as man is to woman" est encodée par l'équation $v_{\text{king}} - v_{\text{queen}} \approx v_{\text{man}} - v_{\text{woman}}$ où v_{x} représente l'embedding du mot x.

Parmi les formalismes présentés, on cite à titre d'exemple l'algorithme Word2vec [1] (Church, 2017) qui traduit l'intuition en un problème d'apprentissage supervisé consistant à prédire un mot à partir de son contexte (CBOW) ou alors de prédire le contexte d'un mot à partir du mot en question (Skip-gram). Un autre formalisme (GloVe [2]) a été proposé par des chercheurs de Stanford (Pennington, Socher, & Manning, 2014), celui-ci repose sur la factorisation d'une matrice de co-occurences.

L'enjeu de notre côté est de transposer cette logique d'embedding construit à partir d'un voisinage à l'univers financier où l'on ne dispose pas d'un vocabulaire bien défini, comme c'est le cas en NLP.

Nous avons défini un nouvel **embedding** qui incarne l'intuition *You shall know an asset by the company it keeps*. Pour ce faire, nous avons défini un voisinage et un formalisme adéquats.

La figure 1.7 met en lumière la transposition de la notion de voisinage de l'univers NLP à l'univers financier. Ainsi, nous sommes en mesure de définir l'embedding en comparant chaque actif aux autres actifs de la même catégorie, i.e en comparant un secteur à ses "voisins" secteurs, une industrie à ses "voisines" industries au sein du même secteur, et enfin un stock à ses "voisins" stocks au sein la même industrie.

1. Nous en proposons une implémentation en suivant le lien suivant : Implémentation de Word2vec
2. Nous en proposons une implémentation en suivant le lien suivant : Implémentation de GloVe

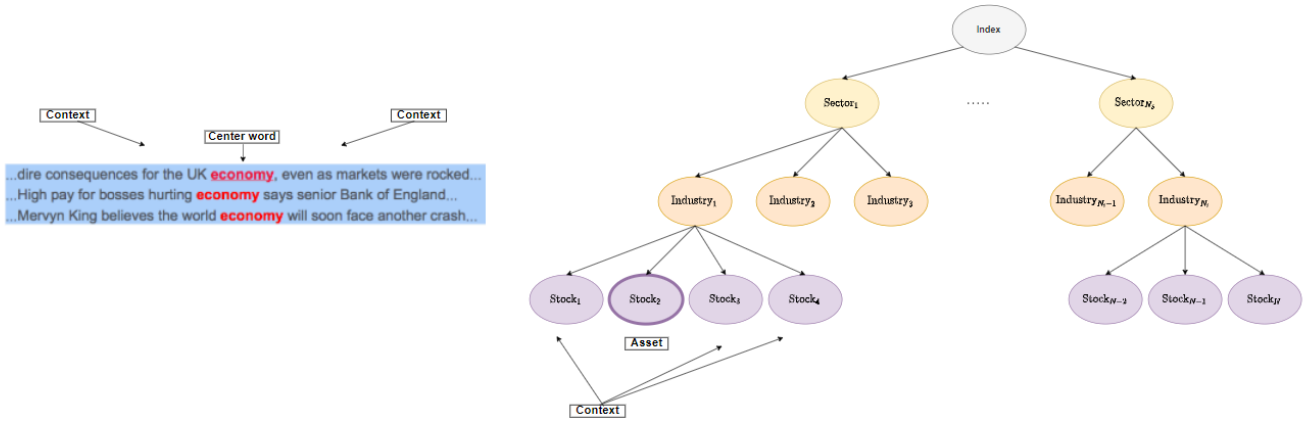


FIGURE 1.7 – Notion de voisinage dans le contexte financier par rapport au contexte du langage

De même qu'en NLP, des formalismes comme le word2vec (Church, 2017) et le GloVe (Pennington et al., 2014) ont permis de créer les embedding à partir de l'intuition des linguistes, nous avons proposé un formalisme à même de traduire l'intuition qu'un actif est défini par ses pairs à l'aide d'un algorithme adéquat.

L'idée fut de s'inspirer de l'algorithme TrueSkill (Herbrich, Minka, & Graepel, 2006) proposé par Microsoft afin de comparer et classer les joueurs d'une équipe. Cet algorithme cherche à estimer le skill d'un joueur à partir d'une série de confrontations avec ses concurrents au sein d'une équipe. A chaque joueur est associée une distribution de skill, supposée gaussienne, que l'on cherche à affiner à l'aide d'un apprentissage bayésien.

Le vecteur skill, représentant la distribution du skill d'un actif, serait ainsi issu des confrontations avec ses pairs (ou voisins).

En combinant les nouveaux embeddings basés sur les skills de chaque actif et les architectures de réseaux de neurones basées sur le concept d'attention, nous avons mis en oeuvre un système de classement des stocks à même de se focaliser sur les skills pertinents. Ce qui ouvre le champ à la création de stratégies de sélection de stocks dont les résultats surperforment le marché, ainsi que les méthodes classiques de momentum.

2 | Introducing Sequential Models with applications in Natural Language Processing

2.1 The Hidden Markov Model

Hidden Markov Models (HMMs) are powerful graphical models used to describe sequential data. They were first introduced in a series of statistical papers by Leonard E. Baum (Baum & Petrie, 1966) and other authors in the second half of the 1960s. Besides their rich mathematical structure, HMMs work very well in a wide range of real world applications. Due to the successful results in Speech Recognition (Rabiner & Juang, 1986), the model has become increasingly popular and has been applied to several other areas such as natural language modeling (Manning & Schutze, 1999). There are also many applications in finance : (S. Kim, Shephard, & Chib, 1998) used HMMs for the analysis of stochastic volatility, in comparison with the traditional GARCH model. In (Nystrup, Madsen, & Lindström, 2017), the authors used the model to infer the hidden states associated with the daily returns in financial markets. Other applications include handwriting recognition (Nag, Wong, & Fallside, 1986) and bioinformatics (Krogh, Brown, Mian, Sjölander, & Haussler, 1994), (Durbin, Eddy, Krogh, & Mitchison, 1998), (Baldi & Brunak, 2001), (Manogaran et al., 2018), (Testa, Hane, Ellwood, & Oliver, 2015).

HMMs can be viewed as a dynamical mixture model. Indeed, the model is a simple generalization of the mixture model. It assumes the existence of latent variables called "hidden states", which are responsible of the observable data. the HMM framework no longer assumes that the hidden states are chosen independently as in vanilla mixture models. It rather assumes the latent states form an unobservable Markov chain.

2.2 Introducing Recurrent Neural Networks

2.2.1 From Hidden Markov Models to Recurrent Neural Networks

Hidden Markov Models (HMM) have tackled the problem of modeling sequences for decades, especially in the context of Speech Recognition (Rabiner & Juang, 1986).

HMM models were extremely popular in the 1980s for their rich mathematical structure and their good performances in a wide range of applications. However, they rely on a Markov assumption on the temporal structure of their hidden states, making the model inefficient for modeling long range dependencies.

On the other hand, Recurrent Neural Networks, with their significantly richer memory and computational

capacity, have attained state of the art performances in applications such as Speech Recognition (Graves, Mohamed, & Hinton, 2013). The following section will introduce Recurrent Neural Networks.

2.2.2 Vanilla Recurrent Neural Networks

Feed-forward neural networks make the assumption that the data is independent and identically distributed (i.i.d). Recurrent Neural Networks (RNNs) (Rumelhart, Hinton, & Williams, 1986) are a family of neural networks capable of processing the data in a sequential way. Figure 2.1 is representation of the vanilla RNN architecture.

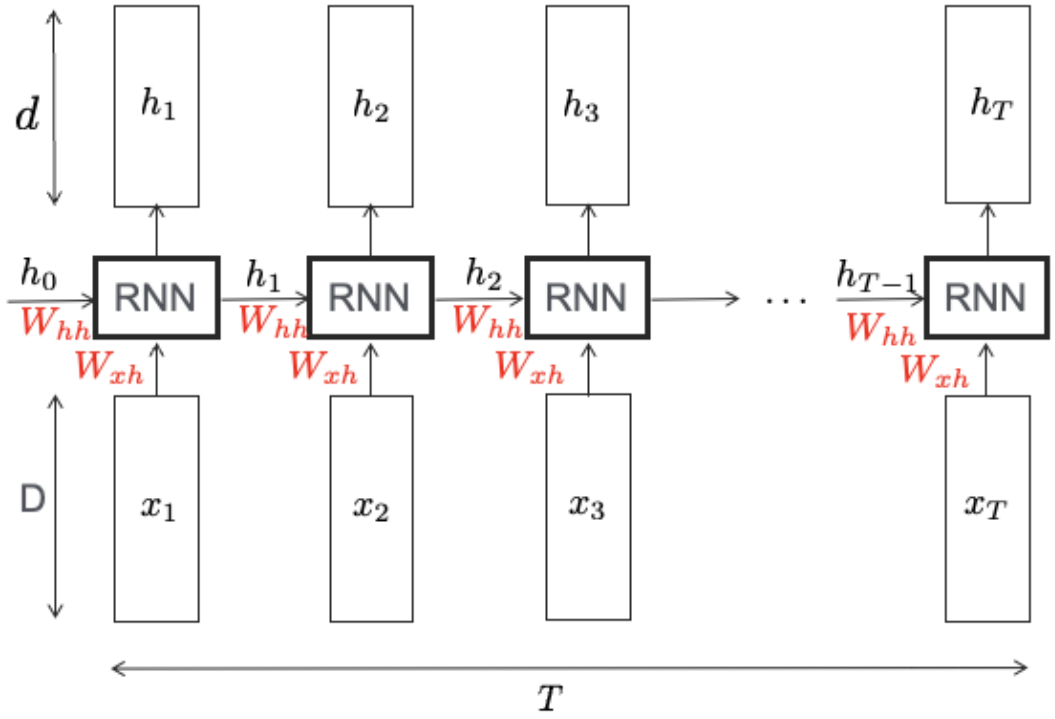


FIGURE 2.1 – Vanilla RNN

The objective is to process an input sequence of D-dimensional vectors $x_1, \ldots, x_T$ sequentially in order to generate a sequence of d-dimensional hidden states $h_1, \ldots, h_T$. The model is parameterized by the two matrices $W_{xh} \in \mathbb{R}^{D \times d}$ and $W_{hh} \in \mathbb{R}^{d \times d}$

At each time step $t \in \{1, \ldots, T\}$, there are two inputs to the hidden layer : the previous hidden state $h_{t-1} \in \mathbb{R}^d$, and the input vector at that time step x_t. The former input vector is processed by the weight matrix W_{hh} and the latter by the weight matrix W_{xh} in order to produce the next hidden state h_t, as shown in equation 2.2.1

$$\forall t \in \{1, \dots, T\} \quad h_t = \tanh\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right) \tag{2.2.1}$$

Unfortunately, it is difficult to access information from many steps back if T is too large due to problems like vanishing or exploding gradients (Bengio, Simard, & Frasconi, 1994).

— **The exploding gradient problem :** This problem refers to the large increase in the gradient norm during training. In order to overcome this issue, Thomas Mikolov introduced a simple heuristic solution called **gradient clipping** in his PhD thesis in 2012. It consists in rescaling the gradient norm, whenever it goes over a threshold. The theoretical justification was then proposed in (Pascanu et al., 2013).

— **The vanishing gradient problem :** (Pascanu et al., 2013) showed that gradient signal from far away is lost because it's a lot smaller than the gradient signal from close-by.

To get the intuition of this issue, let us consider the example of a classification problem using an RNN.

We recall the equation that describes the hidden states transition,

$$\forall t \in \{1, \dots, T\} \quad h_t = \sigma\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right) \tag{2.2.2}$$

Let us consider an output layer for the classification task with K possible outputs :

$$\hat{y}_T = \text{Softmax}\left(W_o^T h_T + b_o\right) \tag{2.2.3}$$

Where $W_o \in \mathbb{R}^{d \times K}$ and $b_o \in \mathbb{R}^K$

Hence, the loss function J we wish to optimize is the cross entropy between the final prediction $\hat{y}_T$ and the true output y.

In order to optimize the loss function, we will use Backpropagation through time as explained in (Werbos, 1990).

For that, we need to calculate $\frac{\partial J(\theta)}{\partial h_t}$ for all $t \in \{1, \dots, T\}$ using the Jacobian matrices $(\frac{\partial h_t}{\partial h_{t-1}})_{2 \leq t \leq T}$ as follows :

$$\forall t \in \{1, \dots, T\} \quad \frac{\partial J(\theta)}{\partial h_t} = \frac{\partial J(\theta)}{\partial h_T} \prod_{t < t' \leq T} \frac{\partial h_{t'}}{\partial h_{t'-1}}$$

We would like to explain the vanishing gradient problem for a simple case (a linear activation function). The proof in the general case can be found in (Pascanu et al., 2013).

1. **Simple case : If σ is the identity function**

   In that case,

$$\begin{aligned} \forall t \in \{1, \dots, T\} \quad \frac{\partial h_t}{\partial h_{t-1}} &= \operatorname{diag}\left(\sigma'\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right)\right) W_{hh} \\ &= I W_{hh} \\ &= W_{hh} \end{aligned}$$

   And consequently, by taking the gradient of the objective function J with respect to the t-th hidden state, we get :

$$\begin{aligned} \forall t \in \{1, \dots, T\} \quad \frac{\partial J(\theta)}{\partial h_t} &= \frac{\partial J(\theta)}{\partial h_T} \prod_{t < t' \leq T} \frac{\partial h_{t'}}{\partial h_{t'-1}} \\ &= \frac{\partial J(\theta)}{\partial h_T} W_{hh}^{T-t} \end{aligned}$$

   The vanishing gradient problem occurs when the matrix W_{hh} is too "small". Let us suppose for example that the eigen values of W_{hh}, denoted $\lambda_1, \dots, \lambda_M$, verify : $\forall i \in \{1, \dots, M\} \quad |\lambda_i| < 1$.

   Then,

$$\forall t \in \{1, \dots, T\} \quad \frac{\partial J(\theta)}{\partial h_t} = \sum_{m=1}^{M} c_m \lambda_m^{T-t} q_m \quad (\text{with } c_1, \dots, c_M \in \mathbb{R})$$

   When $T - t$ is too large, λ_m^{T-t} approaches 0, so the gradient **vanishes**.

2. **Generalization : If σ is a non linear activation function**

   In (Pascanu et al., 2013), the authors generalize the result for the nonlinear activation function σ. They get the same result with a stronger assumption : The module of the eigen values of W_{hh} should be smaller than a γ value, which depends on the dimensionality of the vectors and the activation function σ.

In order to address the vanishing gradient problem, (Pascanu et al., 2013) used a regularization term to ensure that the back-propagated gradients neither increase or decrease in magnitude. The regularization term forces the Jacobian matrices to preserve the norm only in relevant directions.

Beyond the solutions discussed so far, new architectures based on gated activation function have been proposed to capture long-term dependencies. The first architecture of that kind, called Long Short-Term

Memory, was proposed by Hochreiter and Schmidhuber in 1997 (Hochreiter & Schmidhuber, 1997) and then refined in (Gers et al., 2000). Hochreiter proposed an architecture using two gates, namely an input gate and an output gate in the original LSTM paper (Hochreiter & Schmidhuber, 1997), while Gers added a forget gate in (Gers et al., 2000). More recently, Gated recurrent units (Cho et al., 2014) were introduced in the context of Neural Machine Translation using two gates to control the information flow from the previous steps.

2.2.3 Long Short-Term Memory (LSTM)

Before transformers, LSTMs achieved state-of-the-art results in a wide range of tasks, including machine translation (Sutskever, Vinyals, & Le, 2014), (Cho et al., 2014) and (Bahdanau, Cho, & Bengio, 2015), language modeling (Sundermeyer, Schlüter, & Ney, 2012), model identification (Z. Wang et al., 2018), time series prediction (Y. Li & Cao, 2018) and Robot Reinforcement Learning (Bakker et al., 2001)

So far, we have introduced Recurrent Neural Networks, which transition from the hidden state h_{t-1} to h_t using a linear transformation and a point-wise non-linearity, as described in equation 2.2.1.

Here, on step t, there is a **hidden state** h^t and a **cell state** c^t that stores long-term information. The core intuition behind LSTMs is to regulate the information that is removed or added to the cell state through different **gates**.

Let us dive into the concept of gate, as shown in figure 2.2a :

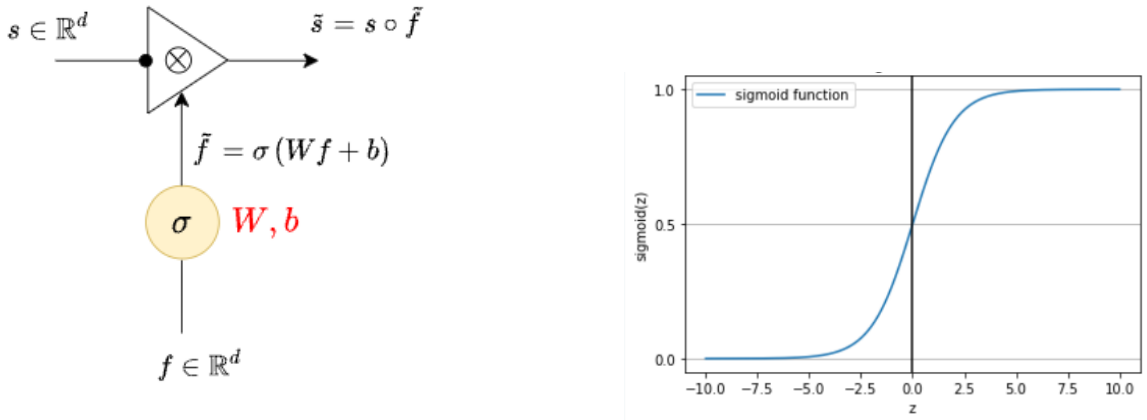
— The signal vector $s \in \mathbb{R}^d$ represents the information that we want to filter using the processing of the vector $f \in \mathbb{R}^d$.

— The processing of the vector f is done through a sigmoid layer, which consists in a linear transformation parameterized by (W, b) and a point-wise non-linearity (using the sigmoid activation function). The output of this layer is the **gate vector** $\tilde{f} := \sigma\left(Wf + b\right)$.

— Figure 2.2b is a representation of the sigmoid function $\sigma : z \mapsto \frac{1}{1+e^{-z}}$. So, the gate vector $\tilde{f}$ is composed of numbers close to zero or close to one, or somewhere in between, describing how much of each dimension (among the d dimensions) we would like to let through.

— The final step is a point-wise multiplication between the signal s and the gate vector $\tilde{f}$. As a result, some dimensions of s are going to be multiplied by values close to 1 in $\tilde{f}$, these dimensions are the ones we would like to keep. On the other hand, some dimensions of s are going to be multiplied by values close to 0, which are the dimensions that need to be updated.

— To get more intuition about this concept, let us suppose that the s signal represents the "memory vector" encoding dimensions like "gender" in the context of language modeling. The gate vector $\tilde{f}$,



(a) Filtering a signal using a sigmoid function and a neural network

(b) The sigmoid function

FIGURE 2.2 – The concept of gate used in LSTMs

by processing a new word, might detect a new "gender". Thus, it will filter the "gender" dimension of the s signal so that it could be replaced with a representation of the new "gender".

As shown in figure 2.3, the LSTM architecture controls the information flow of the cell state using three gates. Let us explain, step by step, how we transition from the couple of the previous hidden and cell states (h^{t-1}, C^{t-1}) to the new couple of hidden and cell states (h^t, C^t) using the "fresh" information x^t.

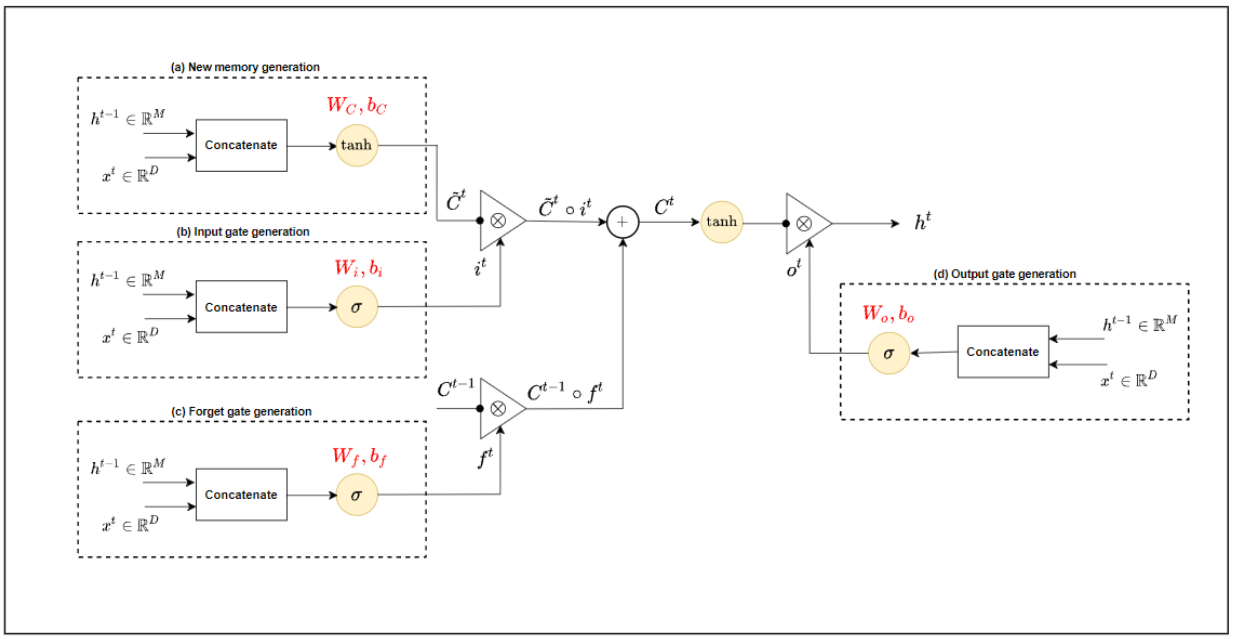


FIGURE 2.3 – The Long Short-Term Memory architecture

— **Step (1) : Generating the new memory candidate**

During this stage, we essentially combine the newly observed vector x^t and the previous hidden

state h^{t-1} to generate a new vector $\tilde{C}^t$ (eq 2.2.4), as shown in figure 2.3 part(a). Therefore, the vector $\tilde{C}^t$ can be seen as a representation of the fresh vector x^t in light of the contextual past. The parameters involved are $W_C \in \mathbb{R}^{(D+M)\times M},\ b_C \in \mathbb{R}^M$. The activation function ($\tanh$) pushes the values between -1 and 1.

$$\tilde{C}^t = \tanh\left(W_C[h^{t-1}, x^t] + b_C\right) \tag{2.2.4}$$

where $[h^{t-1}, x^t] \in \mathbb{R}^{D+M}$ is the concatenation of h^{t-1} and x^t

— **Step (2) : Generating the input gate vector**

As shown in figure 2.3 part(b), we use the x^t vector and h^{t-1} to generate the input gate vector i^t (eq 2.2.5) in order to determine which dimensions of the generated $\tilde{C}^t$ are worth preserving. The parameters involved are $W_i \in \mathbb{R}^{(D+M)\times M},\ b_i \in \mathbb{R}^M$.

$$i^t = \sigma\left(W_i[h^{t-1}, x^t] + b_i\right) \tag{2.2.5}$$

where σ stands for the sigmoid function

— **Step (3) : Filtering the new candidate using the input gate**

We use the input gate vector i^t to filter the new candidate $\tilde{C}^t$, which results in the point-wise multiplication $\tilde{C}^t \circ i^t$ (where $\circ$ stands for the Hadamard product).

— **Step (4) : Generating the forget gate vector**

Similarly to the input gate vector, the vectors h^{t-1} and x^t are used to generate the forget gate vector f^t (eq 2.2.6). This time, instead of assessing the usefulness of the new generated vector $\tilde{C}^t$ dimensions, the forget gate vector is used to filter the dimensions of the previous memory vector C^{t-1} that need to be updated.

$$f^t = \sigma\left(W_f[h^{t-1}, x^t] + b_f\right) \tag{2.2.6}$$

where σ stands for the sigmoid function

— **Step (5) : Filtering the previous cell state using the forget gate**

Similarly to Step (3), we filter the previous cell state C^{t-1} using the forget fate vector f^t, which results in the point-wise multiplication $C^{t-1} \circ f^t$.

— **Step (6) : Getting the final memory state**

At this stage, we combine the advice of the forget gate vector f^t filtering the past memory C^{t-1}, and the advice of the input gate vector i^t filtering the new generated memory $\tilde{C}^t$. The sum of both results produce the final cell state C^t (eq 2.2.7).

$$C^t = i^t \circ \tilde{C}^t + f^t \circ C^{t-1} \tag{2.2.7}$$

— **Step (7) : Generating the output gate vector**

As the hidden states are used as an input to every single gate (forget, input and output gates), the output gate vector o^t makes the assessment regarding what parts of the cell state (the memory state) C^t need to be exposed in the hidden state h^t. This final step decides what parts of the memory C^t need to be present in the hidden states h^t.

$$o^t = \sigma\left(W_o[h^{t-1}, x^t] + b_o\right) \tag{2.2.8}$$

where σ stands for the sigmoid function

— **Step (8) : Getting the final hidden state**

At this stage, we use the o^t vector to gate the point-wise $\tanh$ of the memory vector C^t, which results in the final hidden state h^t (eq 2.2.9). Moreover, at each time step t, we have the possibility to output the hidden state h^t to represent the processing of the first t input vectors $x^1, \ldots, x^t$.

$$h^t = o^t \circ \tanh(C^t) \tag{2.2.9}$$

To sum up, the LSTM architecture can **write**, **erase** or **read** information from the cell state. For each of these actions, the LSTM makes use of gates to decide which dimensions to modify. All the equations involved in the LSTM architecture are then summarized as follows :

— **Erasing** the content of the previous cell state using the **forget gate** $f^t = \sigma\left(W_f[h^{t-1}, x^t] + b_f\right)$, which results in $f^t \circ C^{t-1}$.

— **Writing** new content to the cell state, using the **input gate** $i^t = \sigma\left(W_i[h^{t-1}, x^t] + b_i\right)$, which results in $i^t \circ \tanh(\tilde{C}^t)$

— **Reading** some content from the cell state $C^t = f^t \circ C^{t-1} + i^t \circ \tanh(\tilde{C}^t)$ using the **output gate** $o^t = \sigma\left(W_o[h^{t-1}, x^t] + b_o\right)$, which results in the hidden state $h^t = o^t \circ \tanh(C^t)$

It's worth noticing that all the weight matrices W_C, W_f, W_o, W_i are applied to the concatenation of h^{t-1} and x^t.

Let us consider a couple of parameters $(W, b) \in \{(W_C, b_C), (W_i, b_i), (W_f, b_f), (W_o, b_o)\}$ and an activation function σ (sigmoid or $\tanh$). As $(W, b) \in \mathbb{R}^{(M+D)\times M} \times \mathbb{R}^M$, we can rewrite W as $\begin{bmatrix} U & V \end{bmatrix}$ with $U \in \mathbb{R}^{M\times M}$ and $V \in \mathbb{R}^{D\times M}$, which leads to the following equation :

$$\sigma\left(W[h^{t-1}, x^t] + b\right) = \sigma\left(Uh^{t-1} + Vx^t + b\right) \tag{2.2.10}$$

The LSTM architecture doesn't guarantee to solve the vanishing or exploding gradient problems. However, it ensures the model to learn long-distance dependencies by preserving the information in the cell state. Indeed, the information in a particular dimension is preserved if the forget gate vector and the input gate vector are respectively set to 1 and 0 for this particular dimension. It's much harder for a vanilla RNN to learn weight matrices W_{hh} and W_{xh} that preserve the information in any dimension of the hidden states.

2.2.4 Gated Recurrent Units

The Gated Recurrent Units was introduced in 2014 (Cho et al., 2014) by Kyunghyun Cho. The GRU structure is similar to the LSTM one, except that there isn't any cell state and it only contains 2 gates.

Let us dive into the GRU architecture, represented in figure 2.4

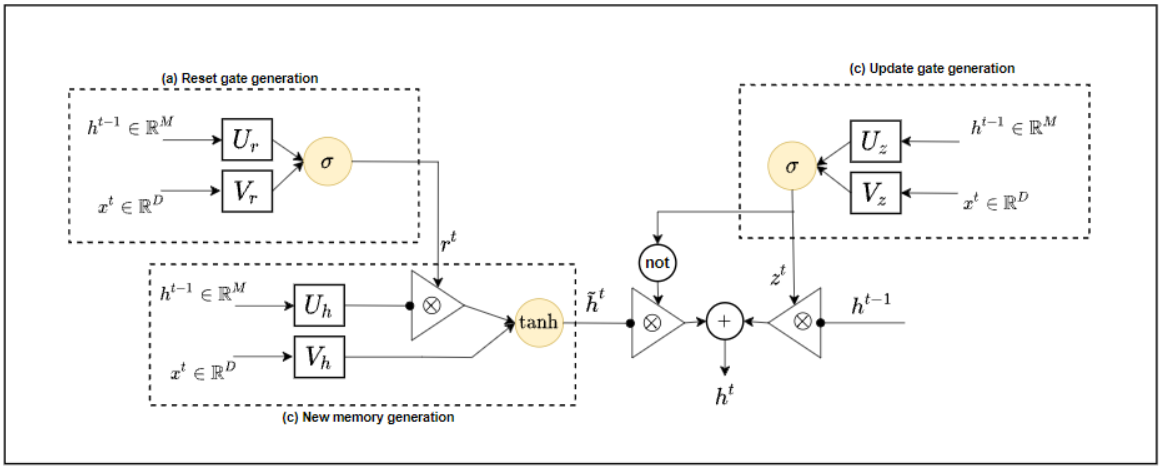


FIGURE 2.4 – The Gated Recurrent Units architecture

— **Step (1) : Generating the reset gate vector**

The objective of the reset gate vector (eq 2.2.11) is to determine which dimensions of the previous

hidden state h^{t-1} are relevant to the computation of the new memory candidate. Similarly to the LSTM gates, it is computed by processing the previous hidden state h^{t-1} and the newly observed vector x^t, in the reduced way described in equation 2.2.10.

$$r^t = \sigma\left(U_r h^{t-1} + V_r x^t\right) \tag{2.2.11}$$

where σ stands for the sigmoid function

— **Step(2) : Generating the new memory candidate**

This stage is analogous to the memory generation in the LSTM architecture. The new candidate $\tilde{h}^t$ is obtained by combining the information from the newly observed vector x^t and the filtered version of the previous hidden state using the reset gate vector r^t. A point-wise $\tanh$ is then applied to push the values between -1 and 1 (eq 2.2.12).

$$\tilde{h}^t = \tanh\left(r^t \circ U_h h^{t-1} + V_h x^t\right) \tag{2.2.12}$$

where $\circ$ stands for the Hadamard product.

— **Step (3) : Generating the update gate vector**

The update gate vector z^t is responsible for balancing between the information from the previous hidden state h^{t-1} and the information from the new memory candidate $\tilde{h}^t$. The vector is calculated using the previous hidden state h^{t-1} and the newly observed vector x^t using the parameters U_z, V_z (eq 2.2.13)

$$z^t = \sigma\left(U_z h^{t-1} + V_z x^t\right) \tag{2.2.13}$$

where σ stands for the sigmoid function

— **Step (4) : Getting the final hidden state**

The final hidden state h^t is generated using the update gate vector z^t. As showed in equation 2.2.14, if a particular dimension of z^t is close to 1, it means that almost all the information from the previous hidden state is copied out to h^t. Conversely, if it's close to 0, it means that the information from $\tilde{h}^t$ should be carried forward to h^t.

$$h^t = z^t \circ h^{t-1} + (1 - z^t) \circ \tilde{h}^t \tag{2.2.14}$$

2.2.5 Different applications of RNNs

With the LSTM or the GRU model, we have discussed how to learn a mapping from the input space $\mathcal{X}$ into the hidden space $\mathcal{H}$.

This framework can support three types of applications, corresponding to the fact that at least one of the spaces $\mathcal{X}$ and $\mathcal{H}$ encodes sequences.

— The **One to Many** application : It consists in learning mapping functions of the form $\Phi_\theta : X \in \mathcal{X} = \mathbb{R}^D \mapsto (h_i^1, \ldots, h_i^T) \in \mathcal{H} = \mathbb{R}^{T \times d}$ using the LSTM/GRU model $\mathcal{E}_\theta$ as shown in figure 2.5. Image captioning (Vinyals, Toshev, Bengio, & Erhan, 2015) is a typical example, where the description of an image is generated. An image is mapped into a feature vector, which in turn becomes the input for an LSTM architecture.

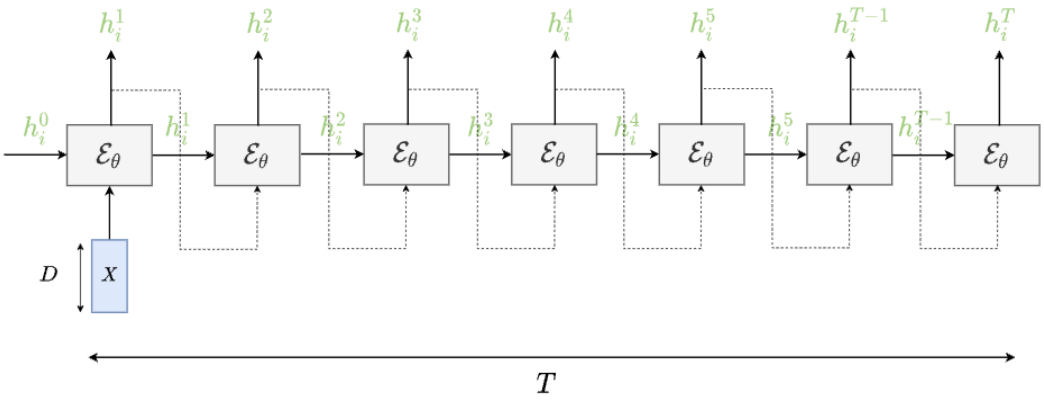


FIGURE 2.5 – The Vector to Sequence framework

— The **Many to One** application : It consists in learning mapping functions of the form $\Phi_\theta : (X_i^1, \ldots, X_i^T) \in \mathcal{X} = \mathbb{R}^{T \times D} \mapsto h_i^T \in \mathcal{H} = \mathbb{R}^d$ using the LSTM/GRU model $\mathcal{E}_\theta$ as shown in figure 2.6. Sentiment Analysis (Murthy, Allu, Andhavarapu, Bagadi, & Belusonti, 2020) is a typical example, where each sentence is mapped to the last hidden hidden state of the LSTM layer. A final Dense layer is then used to predict the sentiment of the sentence (positive, negative or neutral for instance).

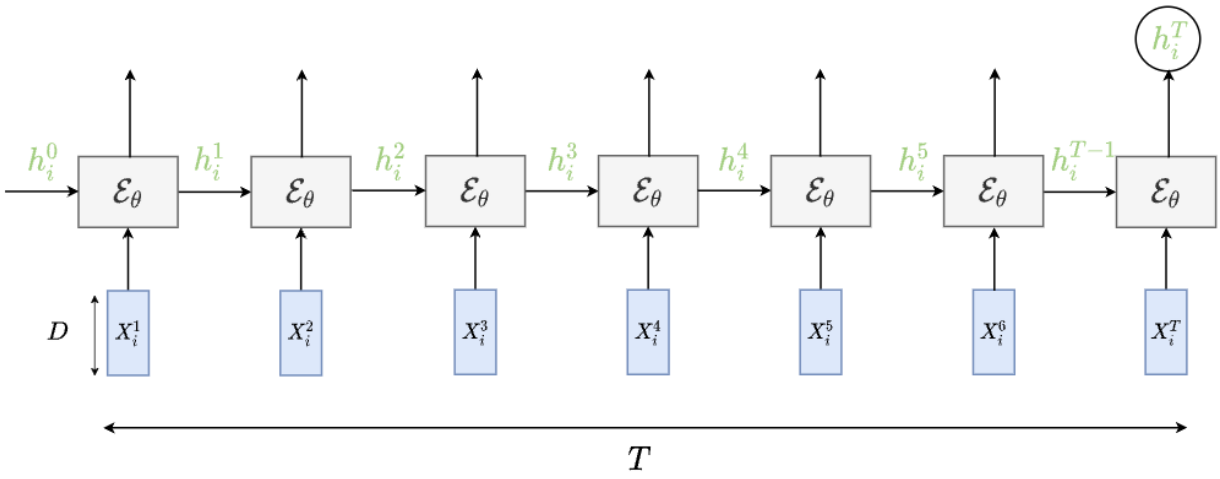


FIGURE 2.6 – The sequence to sequence framework

— The **Many to Many** application : It consists in learning mapping functions of the form $\Phi_\theta : (X_i^1, \ldots, X_i^T) \in \mathcal{X} = \mathbb{R}^{T \times D} \mapsto (h_i^1, \ldots, h_i^T) \in \mathcal{H} = \mathbb{R}^{T \times d}$ using the LSTM/GRU model $\mathcal{E}_\theta$ as shown in figure 2.7. Part of Speech Tagging (P. Wang, Qian, Soong, He, & Zhao, 2015) is a typical example, where the objective is to tag each word of a sentence with its "Part-of-Speech" tag.

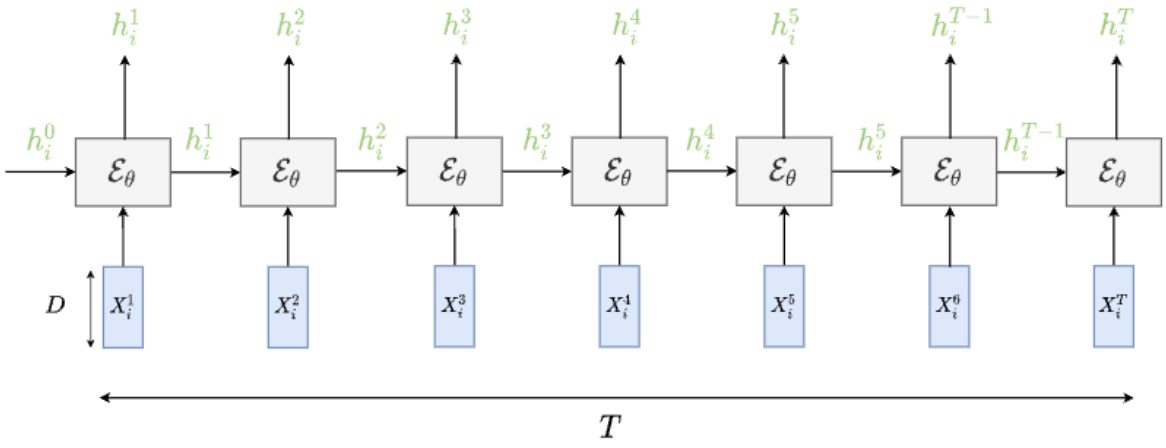


FIGURE 2.7 – The sequence to sequence framework

2.2.6 Applying the RNN/LSTM Model to predict the next word

Word Embeddings

The English Vocabulary contains approximately $V = 13 \times 10^6$ words (or tokens).

The first step in any NLP problem is to map the V tokens into a D-dimensional [1] space encoding all semantics of the language. Each dimension is responsible of some meaning like the gender, the tense,

1. In general, D is between 50 and 300

etc.

We can first represent the tokens as numbers in $\{1, \ldots, V\}$. Each token is then associated with a unique integer $\in \{1, \ldots, V\}$.

Rather than representing the tokens by their indexes, the equivalent representation would be to represent them as vectors of size V with 1 at the index position and zeroes in all the other positions. These vectors are called the **one hot vectors** associated with the tokens.

Predicting the next word using LSTM architectures

For that, we consider a corpus composed of several documents in some language containing V possible words. Using an unsupervised model, we can map each word with a D dimensional vector called **embedding**.

We would like for instance to predict the next word based on the previous T words. Therefore, we organise the dataset in sequences of words that we would like to map to the next words.

We denote each sequence $w_i = (w_i^1, \ldots, w_i^T) \in [V]^T$ (where $[V]$ stands for $\{1, \ldots, V\}$) and the label associated with it $y_i \in [V]$. So, the training data is $\mathcal{T} = \{(w_i, y_i)\}_{i=1}^N$.

The model aims at predicting the next word. Therefore, it's going to map each sequence w_i to a probability distribution $\hat{y}_i \in \Sigma_V$ over all the possible V words.

Where $\Sigma_n := \{y = (y_1, \ldots, y_n) \in [0,1]^n \quad \text{such that} \quad \sum_{i=1}^{n} y_i = 1\}$.

Figure 2.8 shows an example of a sequence w_i "*The students find the course*", processed with a Recurrent Neural Network in order to predict the next word "*interesting*".

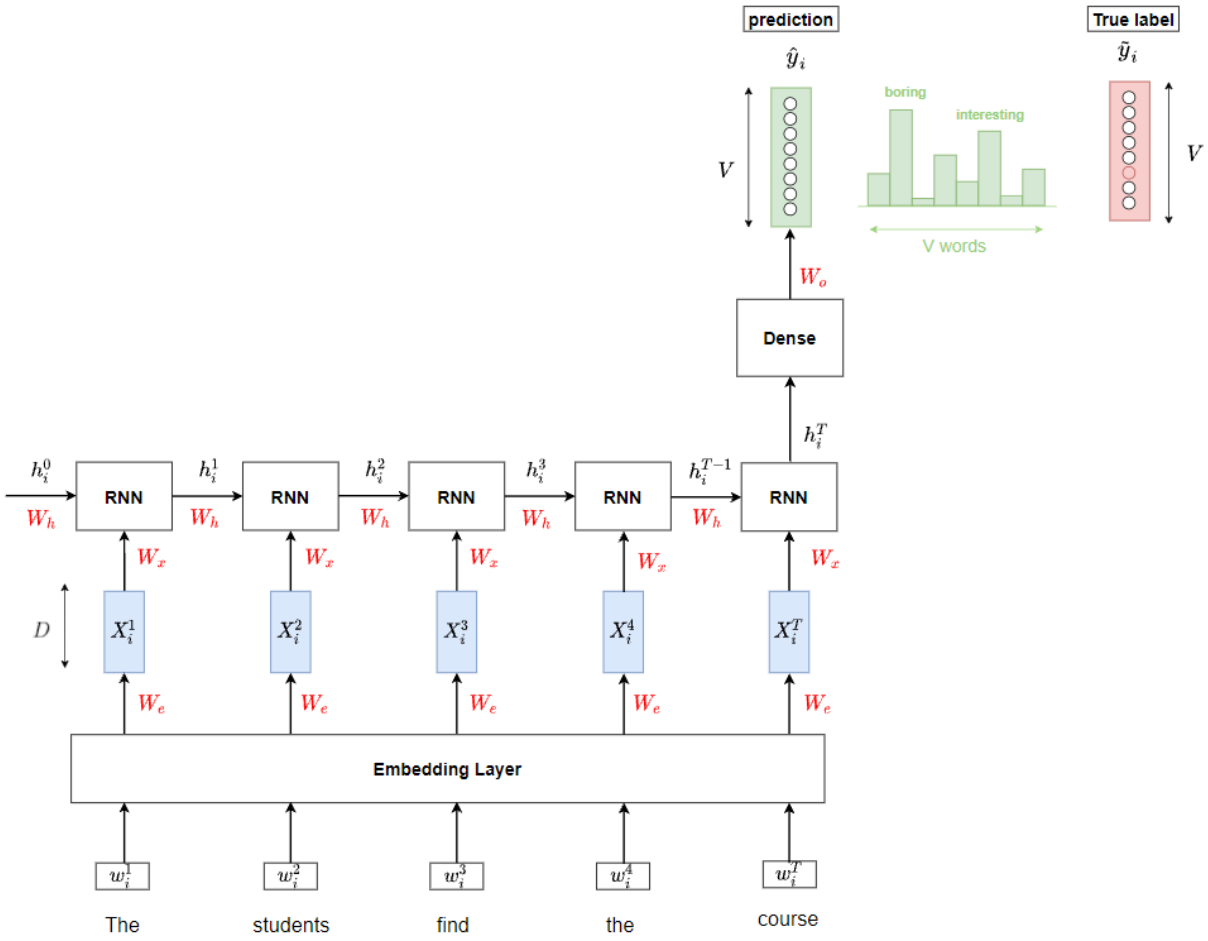


FIGURE 2.8 – RNN for predicting the next word

Below are the details of the prediction process in figure 2.8 :

— First, the input sequence "*The students find the course*" is transformed into a sequence of integers $(w_i^1, \dots, w_i^T)$, via a dictionary that maps each of the V possible words into an index in $[V]$.

— The embedding layer is then used to turn each integer $w_i^t \in [V]$ into a D-dimensional vector representation x_i^t. This is done using an **embedding matrix** W_e of shape (V, D) where each row i represents the embedding of the word of index i. There are different ways of dealing with the embedding matrix :

  — We can consider it as part of the parameters to be trained by backpropagation.

  — We can choose to freeze it by using pretrained word vectors. Such embedding vectors are trained separately on a large corpus of data using a language model.

— The sequence of hidden states $(h_i^1, \dots, h_i^T)$ is then computed sequentially as follows :

$$\forall t \in [T] \quad h_i^t = \sigma\left(W_h h_i^{t-1} + W_x x_i^t\right) \tag{2.2.15}$$

- $x_i^t \in \mathbb{R}^D$ represents the embedding of the new word w_i^t.
- $h_i^{t-1} \in \mathbb{R}^M$ represents the previous hidden state. The size M of the hidden states is a hyper-parameter. We initialize the hidden states with vector of zeros ($h_i^0 = 0_{\mathbb{R}^M}$).
- The RNN layer is parameterized with two matrices : The first weight matrix $W_h \in \mathbb{R}^{M \times M}$ is used to condition the previous hidden state h_i^{t-1}, and the second weight matrix $W_x \in \mathbb{R}^{M \times D}$ is used to condition the new embedding x_i^t.
- σ stands for the non linear activation function. (like the sigmoid, tanh, etc).
- The choice of the activation function is also a hyperparameter.
- It's worth noticing that the **same** matrices W_h and W_x are used throughout the sequential processing.
- The final hidden state $h_i^T \in \mathbb{R}^M$ summarizes the information of the whole sequence in M dimensions.

- The vector h_i^T is then passed into a dense layer, parameterized with a weight matrix $W_o \in \mathbb{R}^{V \times M}$, with a softmax activation function to get the final prediction $\hat{y}_i = (\hat{y}_i^1, \dots, \hat{y}_i^V)$ as follows :

$$\forall v \in [V] \quad \hat{y}_i^v = \frac{e^{[W_o h_i^T]_v}}{\sum_{v'=1}^{V} e^{[W_o h_i^T]_{v'}}}$$

  Where $[W_o h_i^T]_v$ represents the v-th dimension of the V-dimensional vector $W_o h_i^T$
- The final prediction $\hat{y}_i$ is then compared to the true label y_i.
- We usually use **one hot encoding** to represent the discrete random variable Y_i. It consists in encoding Y_i with a random variable $\tilde{Y}_i = (\tilde{Y}_i^1, \dots, \tilde{Y}_i^V)^T$ such that :

$$\forall v \in [V] \quad \tilde{Y}_i^v = 1_{\{Y_i = v\}}$$

We summarize the notations for an element (w_i, y_i) of the training data $\mathcal{T}$ as follows :

Tensor	Space	Definition
w_i^t	$[V]$	t-th word of the input sequence w_i
X_i^t	$\mathbb{R}^D$	t-th embedding vector of word w_i^t
W_h	$\mathbb{R}^{M\times M}$	First weight matrix of the RNN layer
W_x	$\mathbb{R}^{M\times D}$	Second weight matrix of the RNN layer
h_i^t	$\mathbb{R}^M$	t-th hidden state of the RNN layer
W_o	$\mathbb{R}^{V\times M}$	Weight matrix for the dense layer
$\hat{y}_i$	Σ_V	Model prediction
y_i	$[V]$	The true label
$\tilde{y}_i$	$\{0,1\}^V$	The one-hot vector associated with the true label y_i

The whole architecture can be written as follows :

— Let us denote $\theta = (W_e, W_h, W_x, W_o)$ the parameters of the model and f_θ the function that maps each sequence w_i to the prediction $\hat{y}_i$.

— Thus,

$$\hat{y}_i = f_\theta(w_i)$$

— As a result, the distribution of the label Y conditioned on the input sequence W is :

$$Y \mid W = w_i \sim \mathcal{M}\left(1, [f_\theta(w_i)]_1, \ldots, [f_\theta(w_i)]_V\right)$$

Where :

— $\mathcal{M}(1, \pi_1, \ldots, \pi_V)$ stands for the Multinomial distribution, parameterized by $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_V)$

— $[f_\theta(w_i)]_v$ represents the v-th dimension of the V-dimensional vector $\hat{y}_i = f_\theta(w_i)$

The loss function can be derived from the likelihood as follows :

— The likelihood $\mathcal{L}(\theta)$ can then be written as follows :

$$\begin{aligned}
\mathcal{L}(\theta) &= \prod_{i=1}^{N} \mathbb{P}\left(Y = y_i \mid W = w_i\right) \quad \text{(since } (w_i, y_i)_i \text{ are i.i.d)} \\
&= \prod_{i=1}^{N} \mathbb{P}\left(\tilde{Y} = \tilde{y}_i \mid W = w_i\right) \\
&= \prod_{i=1}^{N} \prod_{v=1}^{V} \left[[f_\theta(w_i)]_v\right]^{\tilde{y}_i^v}
\end{aligned}$$

— Instead of maximizing the likelihood, we can minimize the scaled negative log likelihood. Which gives the following loss function :

$$J(\theta) := -\frac{1}{N}\log\left(\mathcal{L}(\theta)\right)$$
$$= -\frac{1}{N}\sum_{i=1}^{N}\sum_{v=1}^{V}\tilde{y}_i^v \log\left([f_\theta(w_i)]_v\right)$$

We can use the exact same framework to predict the next word by replacing the RNN layer with the LSTM layer, as shown in figure 2.9.

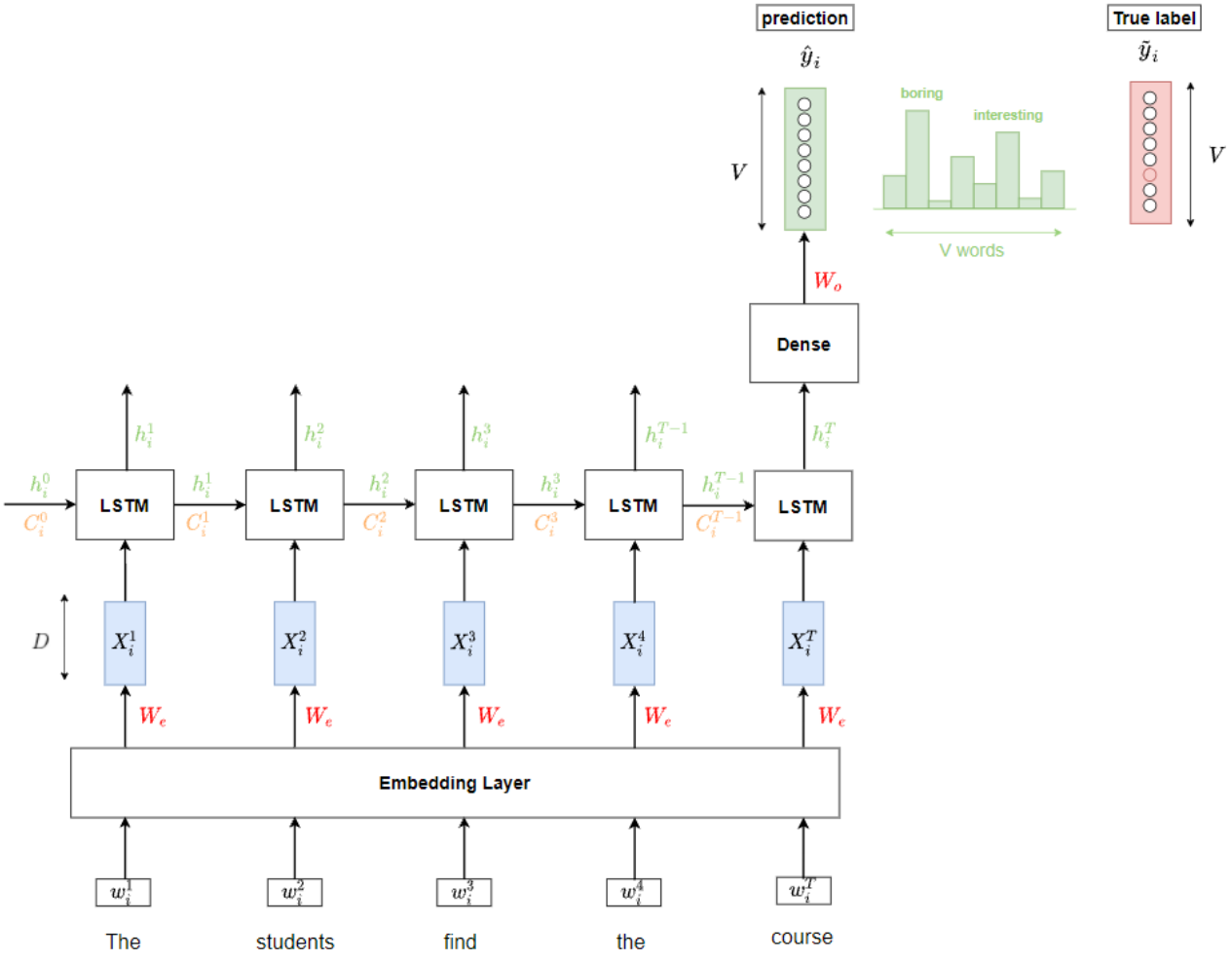


FIGURE 2.9 – Predicting the next word using the LSTM architecture

2.3 The Sequence to Sequence Framework

For Many to Many applications, the LSTM/GRU models can only be applied if the input and the output sequences are of the same length, which can be useful for applications such as POS tagging (P. Wang et al., 2015). However, if we want to learn a mapping Φ_θ from a sequence of input vectors of length T_x into a sequence of output vectors of length T_y (where $T_x \neq T_y$), we need to introduce a new framework, composed of two steps, represented in figure 2.10.

— An encoder layer $\mathcal{E}_{\theta_e}$, parameterized by θ_e learns to map the input sequence $(X_i^1, \ldots, X_i^{T_x}) \in \mathbb{R}^{T_x \times D_x}$ into the sequence of hidden states $h_i^1, \ldots, h_i^{T_x}$.

— The last hidden state $h_i^{T_x}$ is fed into the decoder layer $\mathcal{D}_{\theta_d}$, parameterized by θ_d, as an initial hidden state.

— Let $s_i^0, \ldots, s_i^{T_y}$ be the decoder hidden states. Thus, we have $s_i^0 = h_i^{T_x}$.

— The decoder layer produces the output vectors $s_i^1, \ldots, s_i^{T_y}$.

In the sequence to sequence framework, the encoder and the decoder can be any sequential model such as vanilla RNNs (Kombrink, Mikolov, Karafiát, & Burget, 2011) or LSTMs (Hochreiter & Schmidhuber, 1997).

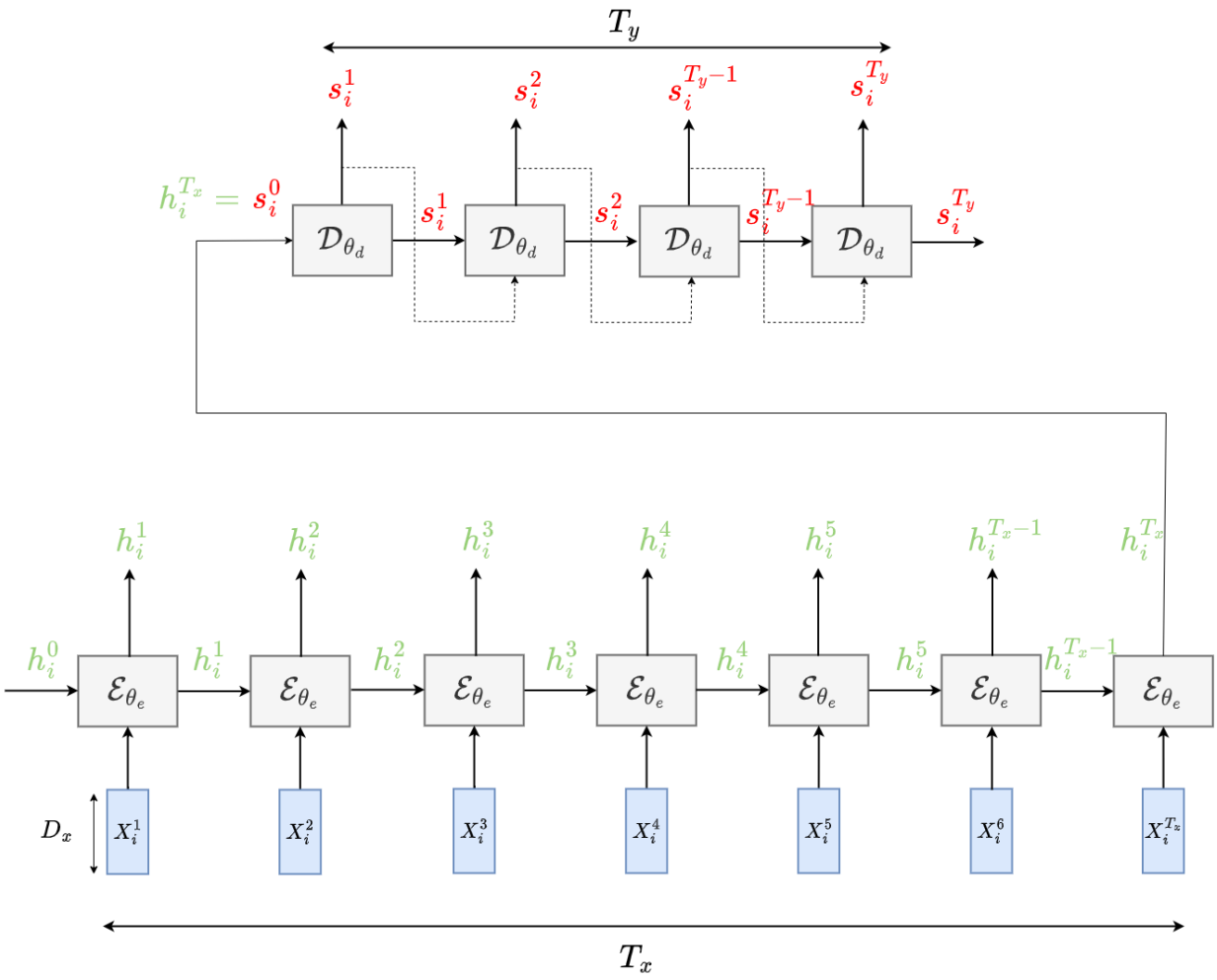


FIGURE 2.10 – The sequence to sequence framework

2.3.1 Applying the Sequence to Sequence Model for Neural Machine Translation

In the previous section, we introduced how we can process a sequence of feature vectors in order to predict a probability distribution in a classification context. The problem we are dealing with is slightly different, in the sense that our objective is to map a sequence of feature vectors into another related sequence. In Natural Language Processing, that would typically be the case for a translation model.

Indeed, the sequence to sequence model was first used for English-French translation (Sutskever et al., 2014).

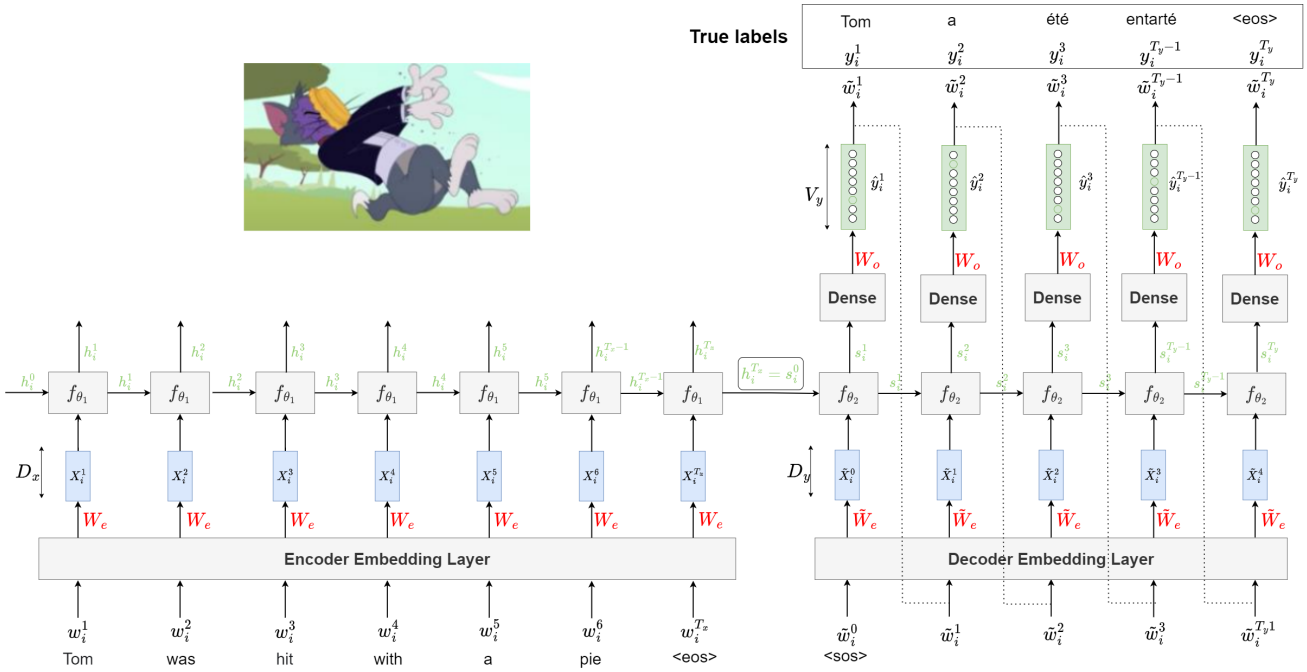


FIGURE 2.11 – The sequence to sequence architecture for machine translation

Figure 2.11 is a representation of such a model.

Basically, the input is a sequence of English words like "Tom was hit with a pie", that we want to translate into the French sentence "Tom a été entarté" using an encoder decoder architecture.

The input data is then a sequence of English words $w_i = (w_i^1, \dots, w_i^{T_x})$ of length T_x, and the target associated with it is a sequence of French words $y_i = (y_i^1, \dots, y_i^{T_y})$ of length T_y.

Such a sequence to sequence model is typically composed of :

— **The encoder** :

The encoder compresses all the information of the input sequence into a fixed length vector as follows :

— As usual, the first step is to map the input words $(w_i^1, \dots, w_i^{T_x})$ into the embedding space of size D_x via the embedding matrix W_e associated with the English vocabulary.

— The embedding vectors $X_i^1, \dots, X_i^{T_x}$ associated with the input words $(w_i^1, \dots, w_i^{T_x})$ are processed using an RNN model f_{θ_1} (the GRU model for instance).

— Let $h_i^t \in \mathbb{R}^M$ represents the hidden state of the encoder at time t :

$$h_i^t = f_{\theta_1}\left(h_i^{t-1}, X_i^t\right)$$

— As the final hidden state $h_i^{T_x}$ encodes all the information in the input sequence, it is going to be used to initialize the hidden states of the decoder.

— **The decoder** :

The decoder is also an RNN which takes the vector $h_i^{T_x}$ and generates an output sequence $(\tilde{w}_i^1, \dots, \tilde{w}_i^{T_y})$.

— Let s_i^t represent the hidden state of the decoder at time t.

— The first hidden state of the decoder s_i^0 is the last hidden state of the encoder $h_i^{T_x}$.

— The first input vector $\tilde{X}_i^0$ is the embedding vector of size D_y associated with the token of index $\tilde{w}_i^0$ "<sos>" (i.e, start of sequence). For that, we use the embedding matrix $\tilde{W}_e$ associated with the French vocabulary.

— Each hidden state s_i^t, at a particular time step $t \in \{1, \dots, T_y\}$, is transformed into a V_y-dimensional discrete distribution $\hat{y}_i^t$ (where V_y is the size of the French vocabulary) via a feed forward neural network parameterized by W_o. Therefore, $\tilde{w}_i^t$ the predicted word at time t is the word whose index corresponds to the highest probability in $\hat{y}_i^t \in V_y$.

— The hidden state s_i^t of the decoder at time $t \in \{2, \dots, T_y\}$ is obtained by processing the previous hidden state s_i^{t-1} and the embedding $\tilde{X}_i^{t-1}$ of the previous predicted word $\tilde{w}_i^{t-1}$

— Let us denote θ the parameters of the model and for all $t \in \{1, \dots, T_y\}$ $\tilde{y}_i^t$ the one hot encoding vector associated with the integer y_i^t.

— Since we have a multiclass classification problem (over V_y possible categories) at each time step $t \in \{1, \dots, T_y\}$, the global loss function $J_i(\theta)$ associated with an input-ouput pair (w_i, y_i) is the average of all the losses $J_i^t(\theta)$ associated with each time step $t \in \{1, \dots, T_y\}$. Consequently :

$$J_i(\theta) = \frac{1}{T_y} \sum_{t=1}^{T_y} \underbrace{\sum_{v=1}^{V_y} \tilde{y}_i^t[v] \log\left(\hat{y}_i^t[v]\right)}_{J_i^t(\theta)} \tag{2.3.16}$$

Where :

— For all $t \in \{1, \dots, T_y\}$ and $v \in \{1, \dots, V_y\}$ $\tilde{y}_i^t[v]$ stands for the v-th dimension of $\tilde{y}_i^t$.

— Similarly, for all $t \in \{1, \dots, T_y\}$ and $v \in \{1, \dots, V_y\}$ $\hat{y}_i^t[v]$ stands for the v-th dimension of $\hat{y}_i^t$.

— **The teacher forcing strategy**

As the decoder predictions are fed back into itself, the model has to predict the whole sequence $\hat{y}_i^1, \dots, \hat{y}_i^{T_y}$ before it compares it to the true sequence $\tilde{y}_i^1, \dots, \tilde{y}_i^{T_y}$ using the loss 2.3.16.

This recursive output-as-input process can result in slow convergence since a bad prediction that occurs at the beginning of the predicted sequence is very likey to affect the rest of the predicted sequence.

An interesting technique that is frequently used in temporal supervised learning tasks (Williams & Zipser, 1989) to overcome this issue is to replace, during the training process, the output vector $\tilde{w}_i^t$ that is fed into the decoder at the next iteration with the true prediction y_i^t as shown in figure 2.12. This technique is called *teacher forcing* since it corrects the predictions at each time step to maximize the chances of producing better next predictions, the same way a teacher would correct the answer of a student one step at a time.

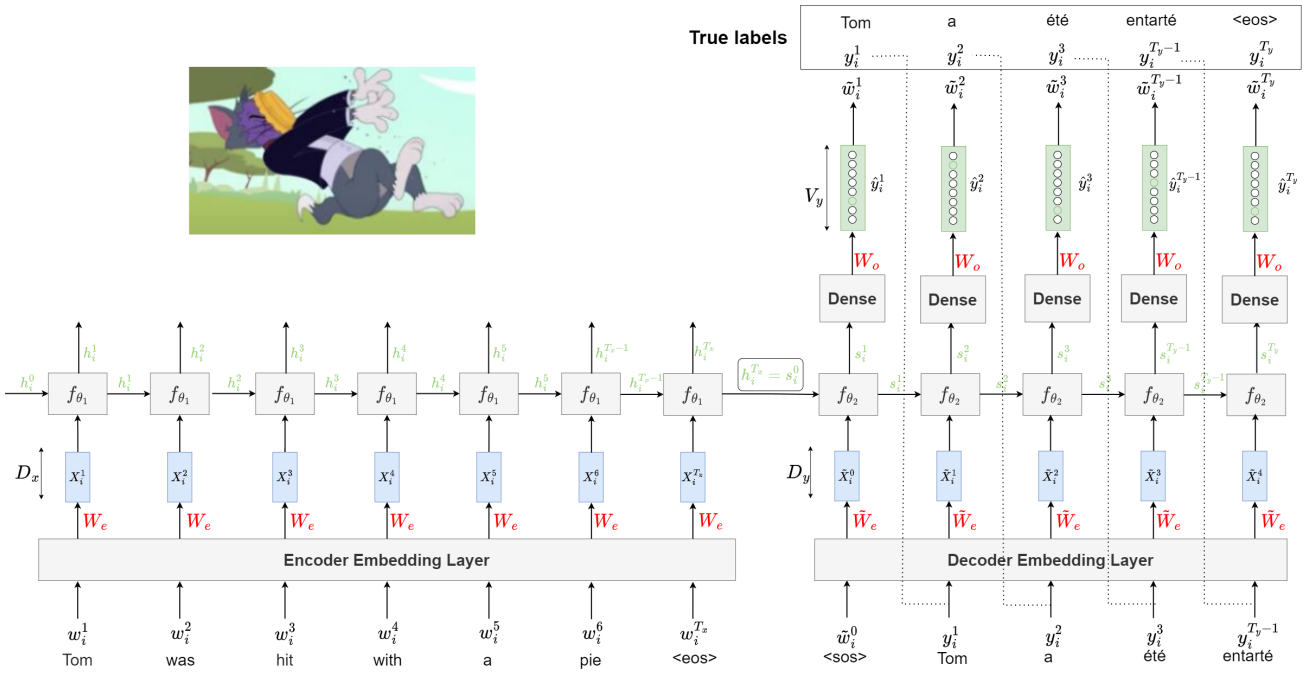


FIGURE 2.12 – The sequence to sequence with teacher forcing architecture for machine translation

2.4 Limitations of classical models

Although Recurrent Neural networks (RNNs) and convolutional neural networks (CNNs) have been sucessfully applied to capture non trivial relationships in complex systems, classic models only perform the task of perception, which consists in learning a mapping between inputs and outputs. They do not carry out sequential reasoning.

Moreover, there are two main challenges with the sequence to sequence framework using RNNs. First, by feeding a single fixed length vector to the decoder, the encoder has to compress all the input information in few dimensions, which leads to a loss of information. Due to this limitation, the performance of the sequence to sequence model degrades rapidly as the length of the input sequence increases.

Second, this architecture doesn't allow **model alignment** between the input and the output sequences. We would like each output sequence to selectively focus on relevant parts of the input sequence.

Let us consider a mapping from a sequence $(X_i^1, \ldots, X_i^{T_x})$ into a sequence $(Y_i^1, \ldots, Y_i^{T_y})$. The intuition of alignment is represented in figure 2.13, it shows how much of each input vector $X_i^{t'}$ should be considered when generating an output vector Y_i^t, for all $(t, t') \in \{1, \ldots, T_y\} \times \{1, \ldots, T_x\}$.

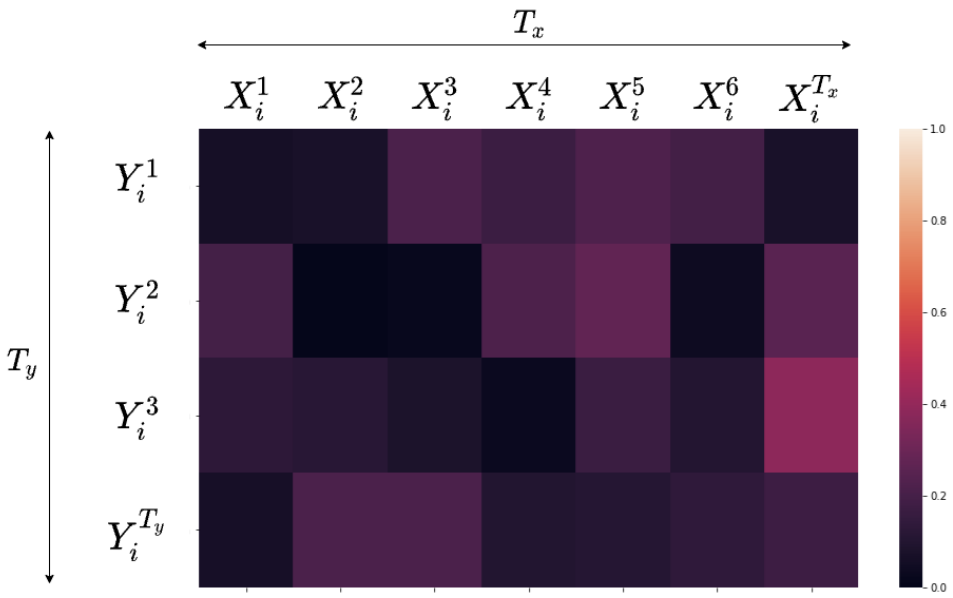


FIGURE 2.13 – Matrix of alignment scores

To sum up, the sequence to sequence framework is not well adapted to modeling long time dependencies and capturing the relevance between the input and the output sequences. Attention mechanisms aim at addressing the aforementioned crucial challenges, especially in applications such as machine translation or time series prediction.

The process of reasoning consists in combining the perception with a selective memory guiding the process of reasoning by focusing on relevant parts of the input or the memory. That's why neuroscientists (Dehaene, 2012), (Deco & Rolls, 2005), (Lindsay, 2020) consider attention as a pillar of learning and reasoning.

2.5 Introducing the Attention Mechanisms in Machine Learning

2.5.1 Query-Retrieval Modeling

Attention mechanisms originate from database Query-Retrieval Problems. Consider the database represented in figure 2.14 where a query is searched through the keys in order to retrieve a value (Garcia-Molina, Ullman, & Widom, 2000).

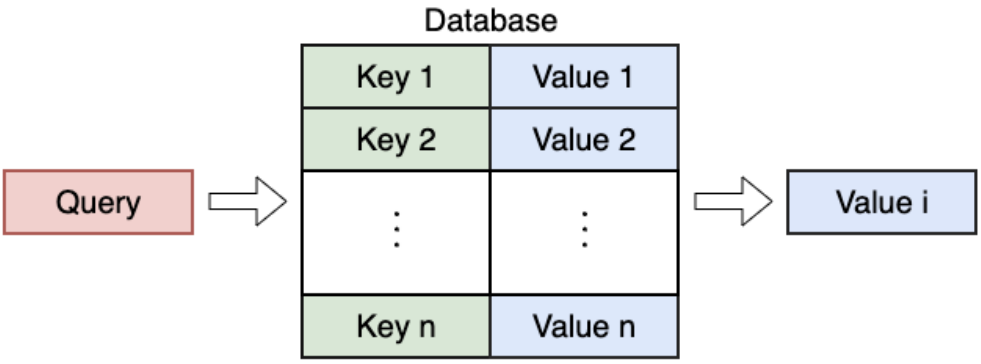


FIGURE 2.14 – Hard Query Retrieval Problem

Attention mechanisms can be viewed as a soft query retrieval process, where multiple keys can correspond to the query, rather than only one. As a result, we need to calculate the similarity between the query and all the keys. The sum of the values, weighted by the computed similarities is the soft-query retrieval vector, or the **attention** vector.

Figure 2.15 represents the different steps involved in calculating the attention vector from a query $q \in \mathbb{R}^{d_q}$, a list of keys $(k_i)_{1 \leq i \leq n} \in \mathbb{R}^{n \times d_k}$ and a list of values $(v_i)_{1 \leq i \leq n} \in \mathbb{R}^{n \times d_v}$

— Using an alignment function a, we calculate the similarity between the query and all the keys as follows :

$$\forall i \in \{1, \ldots, n\} \quad a_i = a(q, k_i)$$

— Several alignment functions have been proposed in the literature in order to get the alignment scores $(a_i)_{1 \leq i \leq n}$, as shown in table 4.1 :

Function	Equation	References
Dot Product	$a(q, k_i) = q^T k_i$	(Luong, Pham, & Manning, s. d.)
Scaled Dot Product	$a(q, k_i) = \frac{q^T k_i}{\sqrt{d_k}}$	(Vaswani et al., 2017)
Luong's Multiplicative al.	$a(q, k_i) = q^T W k_i$	(Luong et al., s. d.)
Bahdanau's Additive al.	$a(q, k_i) = v_a^T \tanh(W_1 q + W_2 k_i)$	(Bahdanau et al., 2017)
Feature-based	$a(q, k_i) = W_{imp}^T \text{act}(W_1 \phi_1(k_i) + W_2 \phi_2(q) + b)$	(Y. Li, Kaiser, Bengio, & Si, 2019)
Kernel Method	$a(q, k_i) = \phi(q)^T \phi(k_i)$	(Yuan et al., 2021)

TABLE 2.1 – Alignment Functions

— The distribution function is used to map the alignment scores $(a_i)_{1 \leq i \leq n}$ to the attention weights $(\alpha_i)_{1 \leq i \leq n}$. The function ensures that $\forall i \in \{1, \ldots, n\} \quad \alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i = 1$.

— We usually use the softmax distribution function in order to get dense alignments as follows :

$$\forall i \in \{1, \ldots, n\} \quad \alpha_i = \frac{e^{a_i}}{\sum_{j=1}^{n} e^{a_j}} \tag{2.5.17}$$

— Sparse alignment [2] can be obtained by using the **sparsemax** (Martins & Astudillo, 2016) or the **sparse entmax** (Martins et al., 2020) distribution functions.

— The attention vector $A\left(q, (k_i)_{1 \leq i \leq n}, (v_i)_{1 \leq i \leq n}\right)$ is then calculated as follows :

$$A\left(q, (k_i)_{1 \leq i \leq n}, (v_i)_{1 \leq i \leq n}\right) = \sum_{i=1}^{n} \alpha_i v_i$$

2. Non zero probabilities are assigned to only a few number of values

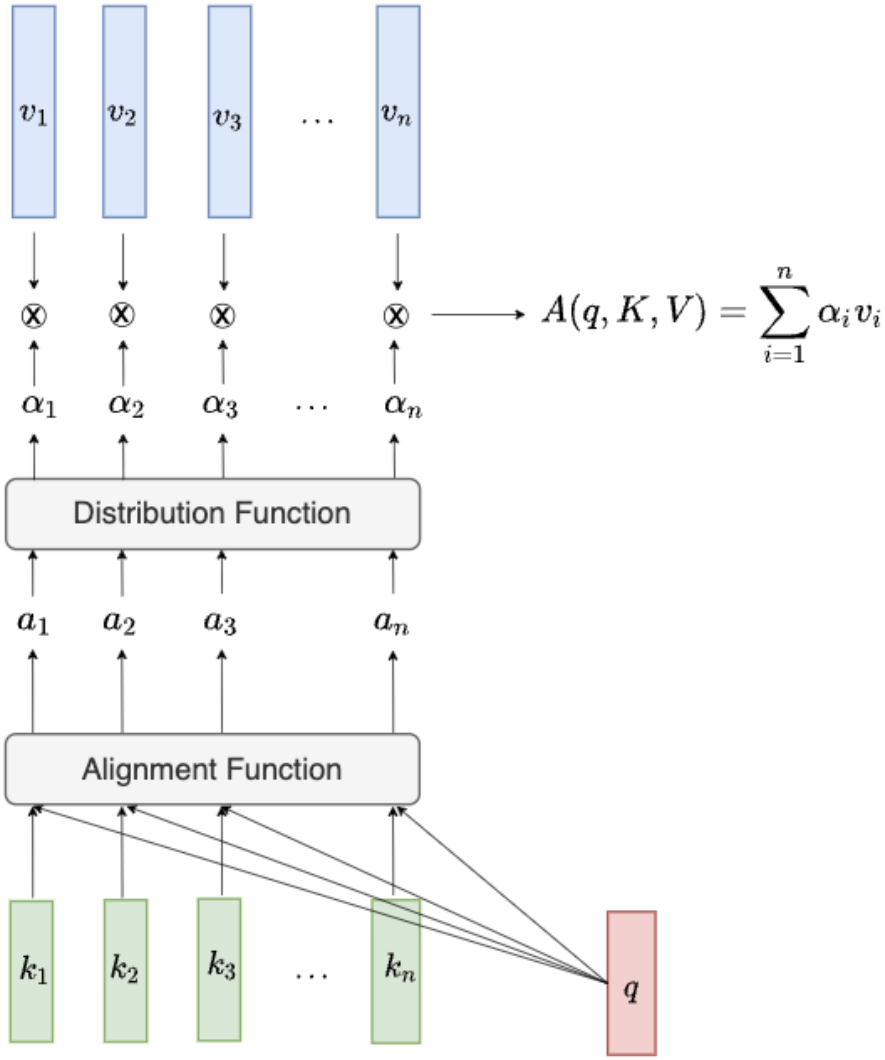


FIGURE 2.15 – Soft-Query Retrieval

2.5.2 Introducing Attention Mechanisms to the Sequence to Sequence framework

The idea of attention mechanisms was first introduced in (Bahdanau et al., 2015) and consists in introducing an attention layer between the encoder and the decoder in order to enable the decoder to decide which part of the encoder outputs are relevant to the generation of the next output.

Formally, let us suppose that our objective is to learn a mapping function Φ_θ from the space of the input sequences $\mathcal{X}$ to the space of the output sequences $\mathcal{Y}$.

Let D_x be the dimensionality of the input vectors and T_x be the length of the input sequences. Let D_y be the dimensionality of the output vectors and T_y be the length of the output sequences.

The training dataset is composed of N input sequences $(X_i^1, \ldots, X_i^{T_x})_{1 \leq i \leq N}$ and N output sequences $(Y_i^1, \ldots, Y_i^{T_y})_{1 \leq i \leq N}$.

We would like to describe the mapping function Φ_θ which transforms an element from the input space $(X_i^1, \ldots, X_i^{T_x}) \in \mathcal{X}$ into an element of the output space $(\hat{Y}_i^1, \ldots, \hat{Y}_i^{T_y}) \in \mathcal{Y}$.

Hence,

$$\mathcal{Y} \ni (\hat{Y}_i^1, \ldots, \hat{Y}_i^{T_y}) = \Phi_\theta(X_i^1, \ldots, X_i^{T_x})$$

The mapping function Φ_θ, described in figure 2.17, can be decomposed into the following transformations :

— An encoder layer $\mathcal{E}_{\theta_e}$ parameterized by θ_e learns to map the input sequence $(X_i^1, \ldots, X_i^{T_x})$ into the sequence of encoder hidden states $h_i^1, \ldots, h_i^{T_x}$.

— A decoder layer $\mathcal{D}_{\theta_d}$ parameterized by θ_d learns to map the encoder hidden states $h_i^1, \ldots, h_i^{T_x}$ to the output sequence $\hat{Y}_i^1, \ldots, \hat{Y}_i^{T_y}$.

— Let $s_i^1, \ldots, s_i^{T_y}$ be the decoder hidden states. An attention layer $\mathcal{A}_{\theta_a}$ parameterized by θ_a learns to assign attention weights $\alpha_i^{<t_y,1>}, \ldots, \alpha_i^{<t_y,T_x>}$ to the encoder hidden states $h_i^1, \ldots, h_i^{T_x}$ when generating the decoder hidden state $s_i^{t_y}$ for all $t_y \in \{1, \ldots, T_y\}$.

— Each decoder hidden state $s_i^{t_y}$ is then mapped into the prediction vector $\hat{Y}_i^{t_y}$ for all $t_y \in \{1, \ldots, T_y\}$ using the final layer $\mathcal{F}_{\theta_f}$.

1. **The Encoder :**

   The encoder $\mathcal{E}_{\theta_e}$ is a GRU model 2.4, parameterized by the following parameters : $U_r \in \mathbb{R}^{M \times M}, V_r \in \mathbb{R}^{D \times M}, U_h \in \mathbb{R}^{M \times M}, V_h \in \mathbb{R}^{D \times M}, U_z \in \mathbb{R}^{M \times M}, V_z \in \mathbb{R}^{D \times M}$

   The GRU maps the input sequence $(X_i^1, \ldots, X_i^{T_x})$ into the sequence $(h_i^1, \ldots, h_i^{T_x})$.

   Let us consider $t_x \in \{1, \ldots, T_x\}$. Equations 2.5.18, 2.5.19 , 2.5.20 , 2.5.21 describe how the encoder $\mathcal{E}_{\theta_e}$ generates the hidden state $h_i^{t_x}$ from $h_i^{t_x-1}$ and $X_i^{t_x}$:

$$r_i^{t_x} = \sigma\left(U_r h_i^{t_x-1} + V_r X_i^{t_x}\right) \quad \text{(Generating the reset gate vector)} \tag{2.5.18}$$

$$\tilde{h}_i^{t_x} = \tanh\left(r_i^{t_x} \circ U_h h_i^{t_x-1} + V_h X_i^{t_x}\right) \quad \text{(Generating the new memory candidate)} \tag{2.5.19}$$

$$z_i^{t_x} = \sigma\left(U_z h_i^{t_x-1} + V_z X_i^{t_x}\right) \quad \text{(Generating the update gate vector)} \tag{2.5.20}$$

$$h_i^{t_x} = (1 - z_i^{t_x}) \circ \tilde{h}_i^{t_x} + z_i^{t_x} \circ h_i^{t_x-1} \quad \text{(Generating the new hidden state)} \tag{2.5.21}$$

2. **The Decoder :**

Let us now consider $t_y \in \{1, \dots, T_y\}$. We would like to generate the decoder hidden state $s_i^{t_y}$ from the previous hidden state $s_i^{t_y-1}$ and a context vector $c_i^{t_y}$ resulting from an attention mechanism applied on the encoder hidden states $h_i^1, \dots, h_i^{T_x}$.

3. **The Attention Layer :**

   The attention layer $\mathcal{A}_{\theta_a}$, represented in figure 2.16, assigns a weight to each encoder hidden state in order to output the final **context vector** c_i^t as follows :

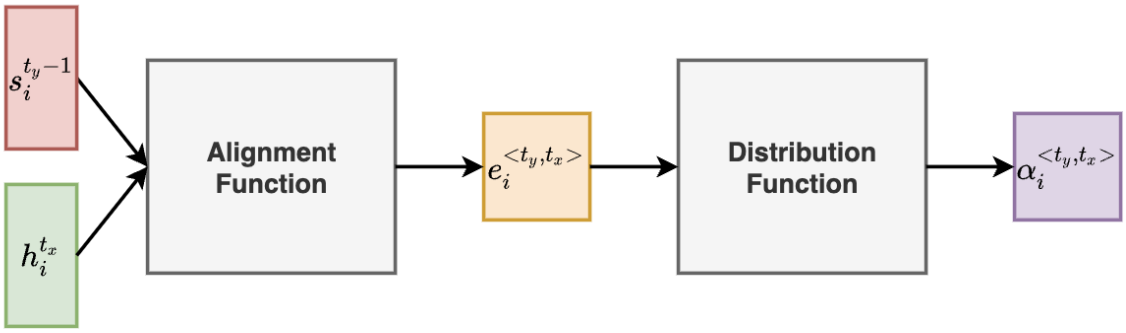


FIGURE 2.16 – The attention layer

— A Feed Forward neural network is applied to calculate an alignment score between the decoder previous hidden state $s_i^{t_y-1}$ and each encoder hidden state $h_i^{t_x} \in \{h_i^1, \dots, h_i^{T_x}\}$. There are multiple alignment functions available, including the additive alignment function of Bahdanau (Bahdanau et al., 2017), as well as the multiplicative alignment function of Luong (Luong et al., s. d.).

$$e_i^{<t_y,t_x>} = \begin{cases} v_a^T \tanh\left(W_1 s_i^{t_y-1} + W_2 h_i^{t_x}\right) & \text{Bahdanau's additive alignment function} \\ s_i^{t_y-1^T} W h_i^{t_x} & \text{Luong's multiplicative alignment function} \end{cases}$$

— The resulting alignment score $e_i^{<t_y,t_x>}$ can then be turned into the attention weight $\alpha_i^{<t_y,t_x>}$ using a distribution function (such as the softmax, the sparsemax (Martins & Astudillo, 2016) or the sparse entmax (Martins et al., 2020) distribution functions).

— The attention vector (also called the context vector) is then calculated as follows :

$$c_i^{t_y} = \sum_{t_x=1}^{T_x} \alpha_i^{<t_y,t_x>} h_i^{t_x}$$

— As a result, the context vector can be seen as the attention vector associated with the query

$q = s_i^{t_y-1}$, the keys $(k_t)_{1\leq t\leq T_x} = (h_i^{t_x})_{1\leq t_x\leq T_x}$ and the values $(v_t)_{1\leq t\leq T_x} = (h_i^{t_x})_{1\leq t_x\leq T_x}$.

$$A\left(q, (k_i)_{1\leq i\leq n}, (v_i)_{1\leq i\leq n}\right) = \sum_{t_x=1}^{T_x} \alpha_i^{<t_y,t_x>} h_i^{t_x} = c_i^{t_y}$$

4. **The Final Layer :**

   The final layer $\mathcal{F}_{\theta_f}$ is then applied on each decoder hidden state $s_i^{t_y}$ in order to generate the prediction $\hat{Y}_i^{t_y}$. The nature of the final layer depends on the kind of application we are dealing with. For instance, in a sentiment analysis problem, the final layer is a basic Dense layer parameterized by $\theta_f = (W_f, b_f)$ with a softmax activation function :

$$\hat{Y}_i^{t_y} = \text{softmax}\left(W_f^T s_i^{t_y} + b_f\right)$$

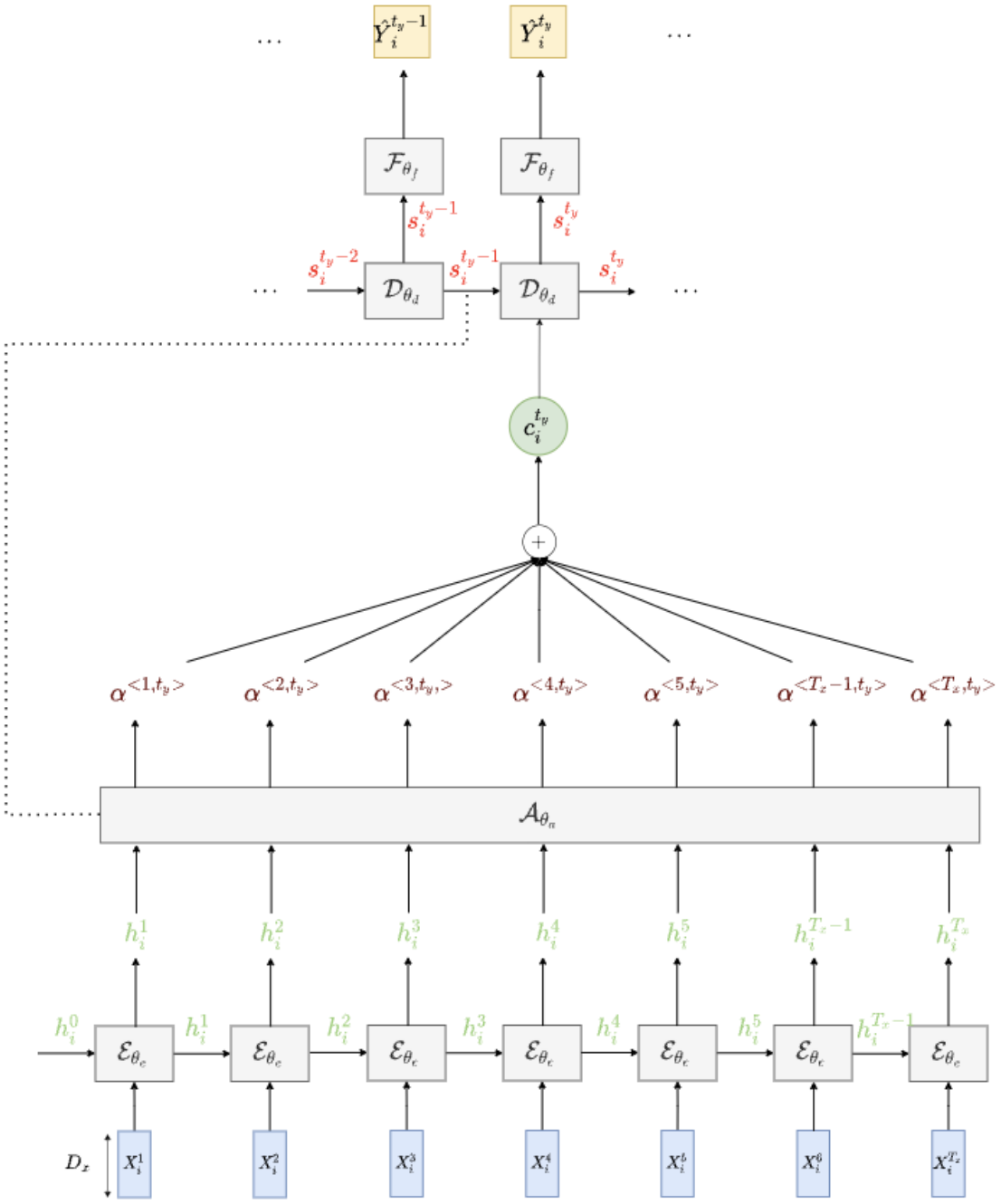


FIGURE 2.17 – Sequence to sequence model with attention

2.5.3 Applying the Sequence to Sequence Model for Neural Machine Translation

By replacing the sequence to sequence architecture represented in 2.11 with the sequence to sequence with attention mechanisms framework defined in 2.18, we obtain the following architecture for Neural Machine Translation.

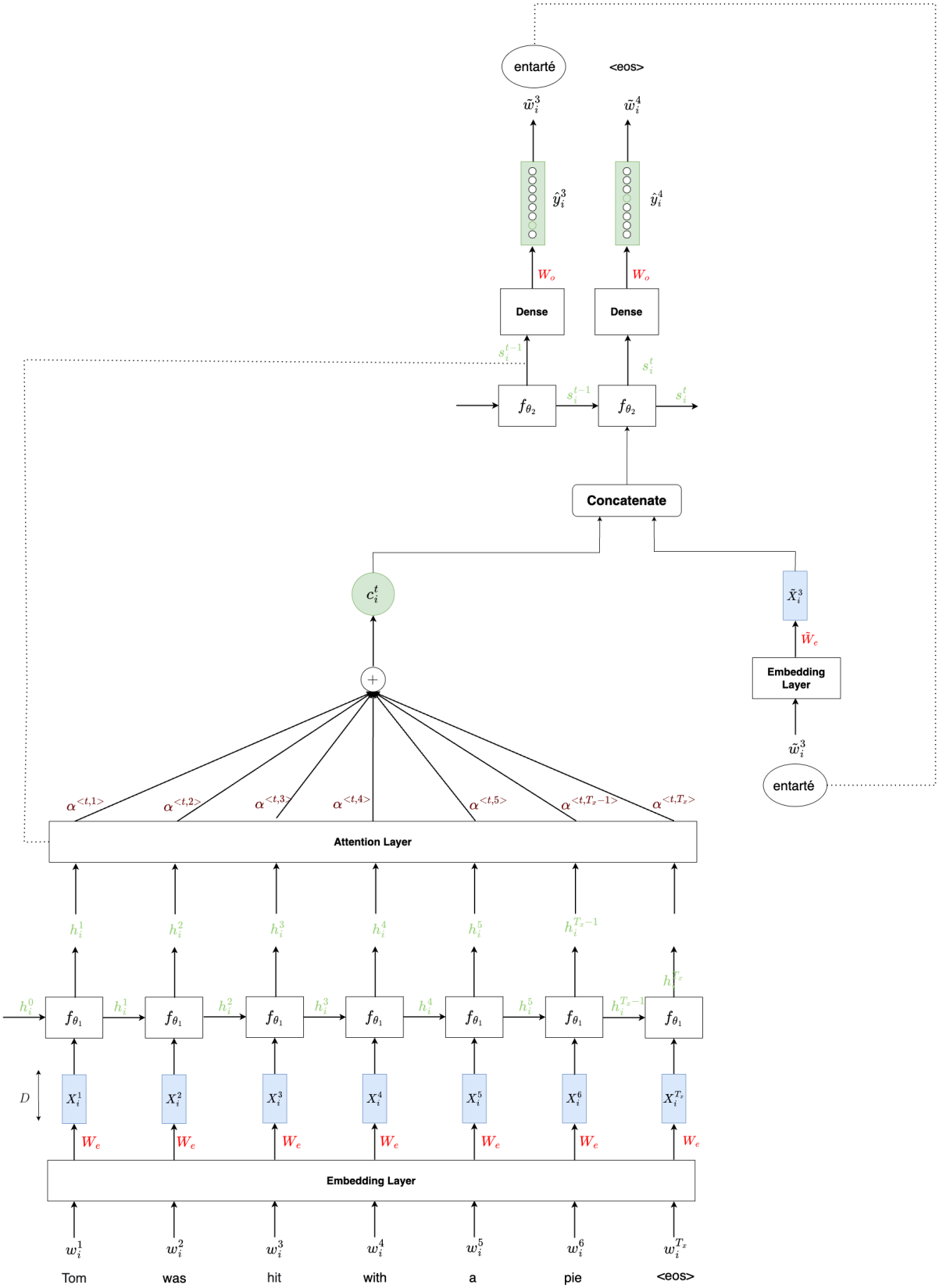


FIGURE 2.18 – The sequence to sequence architecture with attention mechanisms

2.6 The Transformer architecture

2.6.1 Introduction

"Attention is All You Need" (Vaswani et al., 2017) stands out among the most important and interesting papers of the recent years. It presented several improvements to the soft attention algorithm and made it possible to perform the sequence to sequence modeling without having to use recurrent network units.

Therefore, the **Transformer** model proposed in (Vaswani et al., 2017) and represented in the figure 2.19 is entirely based on self-attention mechanisms without using sequence-aligned recurrent architectures.

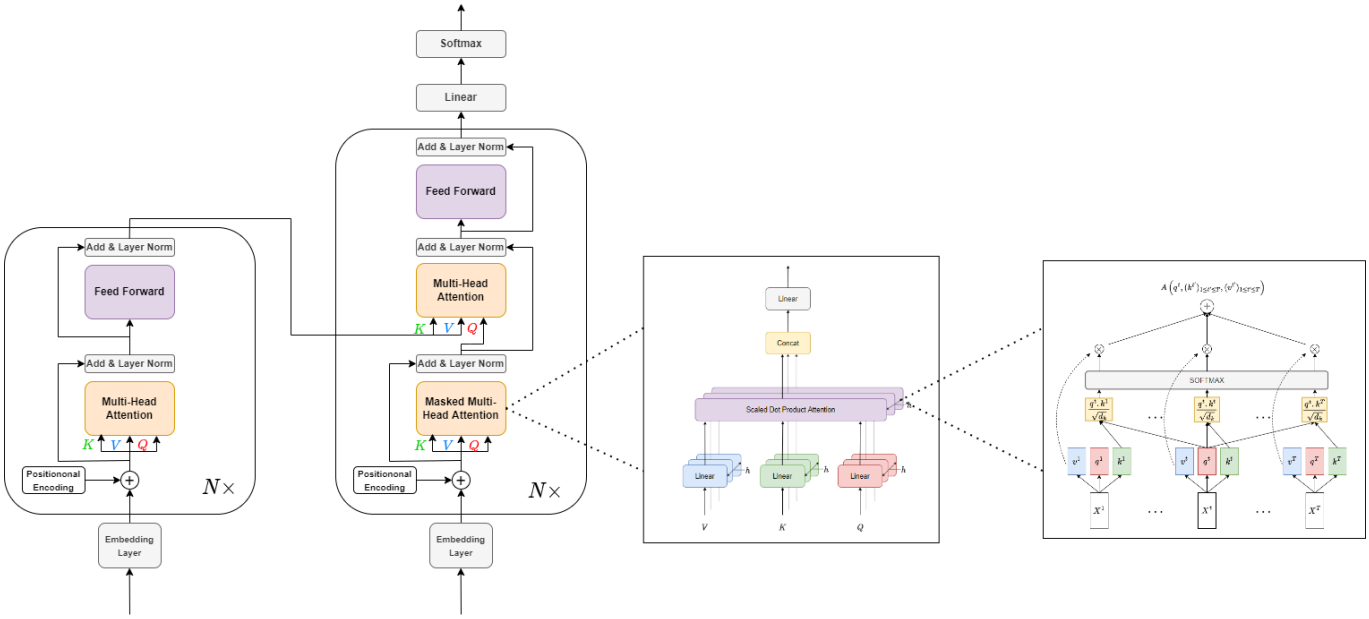


FIGURE 2.19 – The Transformer Architecture

2.6.2 Creating a contextual embedding with Self Attention

Let us consider a sequence of D-dimensional input vectors $(X^t)_{1\leq t\leq T}$. In order to use the attention mechanism, we define the projections of the embeddings X^t onto the d_q-dimensional query space, d_k-dimensional key space and d_v-dimensional value space :

$$\mathbb{R}^{d_q} \ni q^t = W_Q^T X^t$$
$$\mathbb{R}^{d_k} \ni k^t = W_K^T X^t$$
$$\mathbb{R}^{d_v} \ni v^t = W_V^T X^t$$

Where $W_Q \in \mathbb{R}^{D\times d_q}$, $W_K \in \mathbb{R}^{D\times d_k}$ and $W_V \in \mathbb{R}^{D\times d_v}$ are the projection matrices onto the low dimensional query, key and value spaces, respectively. We also need $d_q = d_k$.

Let us consider a query $q^t \in \{q^1, \ldots, q^T\}$. The Self Attention transformation, represented in figure 2.20 consists in creating a contextual embedding $A\left(q^t, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$ associated with the embedding vector X^t.

To that end, the scaled dot product alignment function (Vaswani et al., 2017) is used to calculate the similarity $e^{<t,t'>}$ between the query q^t and the keys $(k^{t'})_{1\leq t'\leq T}$ as follows :

$$e^{<t,t'>} = \frac{q^t.k^{t'}}{\sqrt{d_k}} \tag{2.6.22}$$

A Softmax distribution function is then used to turn the similarity scores $e^{<t,t'>}$ into attention weights $\alpha^{<t,t'>}$ representing the contribution of the embedding $X^{t'}$ in the process of generating the contextual embedding $A\left(q^t, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$.

$$\alpha^{<t,t'>} = \frac{e^{<t,t'>}}{\sum_{s=1}^{T} e^{<t,s>}} \tag{2.6.23}$$

The contextual embedding $A\left(q^t, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$ can then be computed as follows :

$$A\left(q^t, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right) = \sum_{t'=1}^{T} \alpha^{<t,t'>} v^{t'} \tag{2.6.24}$$

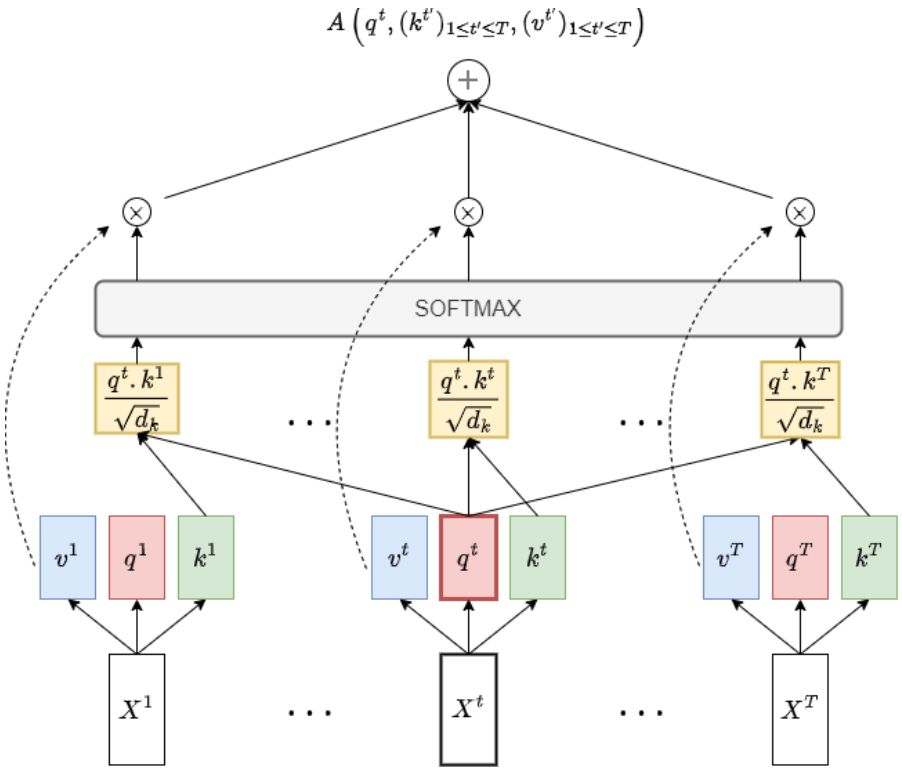


FIGURE 2.20 – Creating the contextual embedding with Self Attention

2.6.3 The Matrix of contextual embeddings

We can generalize the way to create the contextual embedding $A\left(q^t,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right)$ associated with the embedding X^t to all the embedding vectors $(X^{t'})_{1\leq t'\leq T}$.

We consider the following matrix notations :

$$Q=\begin{bmatrix}— & q^1 & — \\ \vdots & \vdots & \vdots \\ — & q^T & —\end{bmatrix}\in\mathbb{R}^{T\times d_q},\quad K=\begin{bmatrix}— & k^1 & — \\ \vdots & \vdots & \vdots \\ — & k^T & —\end{bmatrix}\in\mathbb{R}^{T\times d_k},\quad V=\begin{bmatrix}— & v^1 & — \\ \vdots & \vdots & \vdots \\ — & v^T & —\end{bmatrix}\in\mathbb{R}^{T\times d_v}$$

We define the scaled dot product attention matrix, denoted $A(Q,K,V)$, as follows :

$$A(Q,K,V):=\text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where the notation Softmax(M) for a matrix $M\in\mathbb{R}^{T\times d}$ refers to the Softmax applied to each row of the matrix M.

We have :

$$\begin{aligned}
\text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V &= \text{Softmax}\left(\left[\frac{q^t.k^{t'}}{\sqrt{d_k}}\right]_{\substack{1\leq t\leq T\\1\leq t'\leq T}}\right)V \\
&= \text{Softmax}\left(\left[e^{<t,t'>}\right]_{\substack{1\leq t\leq T\\1\leq t'\leq T}}\right)V \\
&= \left[\alpha^{<t,t'>}\right]_{\substack{1\leq t\leq T\\1\leq t'\leq T}}\begin{bmatrix} — & v^1 & — \\ \vdots & \vdots & \vdots \\ — & v^T & — \end{bmatrix} \\
&= \begin{bmatrix} — & \sum_{t'=1}^{T}\alpha^{<1,t'>}v^{t'} & — \\ \vdots & \vdots & \vdots \\ — & \sum_{t'=1}^{T}\alpha^{<t,t'>}v^{t'} & — \\ \vdots & \vdots & \vdots \\ — & \sum_{t'=1}^{T}\alpha^{<T,t'>}v^{t'} & — \end{bmatrix} \\
&= \begin{bmatrix} — & A\left(q^1,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^t,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^T,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \end{bmatrix}
\end{aligned}$$

Therefore :

$$A(Q,K,V) = \begin{bmatrix} — & A\left(q^1,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^t,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^T,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \end{bmatrix}$$

In other words, the t-th row of the scaled dot product attention matrix $A(Q,K,V)$ is the contextual embedding vectors $A\left(q^t,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right)$ associated with the embedding vector X^t.

2.6.4 MultiHead Attention with the Scaled Dot Product Attention

The scaled dot product attention can be generalized to any query $Q \in \mathbb{R}^{T_q \times d_q}$, key $K \in \mathbb{R}^{T_k \times d_k}$ and value $V \in \mathbb{R}^{T_v \times d_v}$ matrices such that $d_q = d_k$ and $T_k = T_v = T'$.

$$Q = \begin{bmatrix} — & q^1 & — \\ \vdots & \vdots & \vdots \\ — & q^{T_q} & — \end{bmatrix} \in \mathbb{R}^{T_q \times d_q}, \quad K = \begin{bmatrix} — & k^1 & — \\ \vdots & \vdots & \vdots \\ — & k^{T_k} & — \end{bmatrix} \in \mathbb{R}^{T_k \times d_k}, \quad V = \begin{bmatrix} — & v^1 & — \\ \vdots & \vdots & \vdots \\ — & v^{T_v} & — \end{bmatrix} \in \mathbb{R}^{T_v \times d_v}$$

The scaled dot product attention matrix is then defined as follows :

$$A(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V = \begin{bmatrix} — & A\left(q^1, (k^{t'})_{1 \leq t' \leq T'}, (v^{t'})_{1 \leq t' \leq T'}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^t, (k^{t'})_{1 \leq t' \leq T'}, (v^{t'})_{1 \leq t' \leq T'}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^{T_q}, (k^{t'})_{1 \leq t' \leq T'}, (v^{t'})_{1 \leq t' \leq T'}\right) & — \end{bmatrix} \in \mathbb{R}^{T_q \times d_v}$$

The MultiHead Attention module, represented in figure 2.21, consists in applying the attention mechanism defined in the previous section h times in order to capture different notions of similarity.

Hence, for each head $h' \in \{1 \ldots, h\}$, let $W_Q^{h'} \in \mathbb{R}^{d_q \times p_q}$, $W_K^{h'} \in \mathbb{R}^{d_k \times p_k}$ and $W_V^{h'} \in \mathbb{R}^{d_v \times p_v}$ be the h'-th projection matrices of Q, K and V onto the low dimensional key, query and value spaces of size p_q, p_k and p_v, respectively. (We also have $p_q = p_k$).

The h'-th attention head is then defined as $A\left(QW_Q^{h'}, KW_K^{h'}, VW_V^{h'}\right)$.

The concatenation layer aims at stacking all the attention heads together to make a new flatten vector, which is then projected onto $\mathbb{R}^{p_o}$ using the projection matrix $W_o \in \mathbb{R}^{hp_v \times p_o}$. The output of the MultiHead Attention layer P is then computed as follows :

$$P = \text{concat}\left(A\left(QW_Q^1, KW_K^1, VW_V^1\right), \ldots, A\left(QW_Q^h, KW_K^h, VW_V^h\right)\right) W_o \in \mathbb{R}^{T_q \times p_o}$$

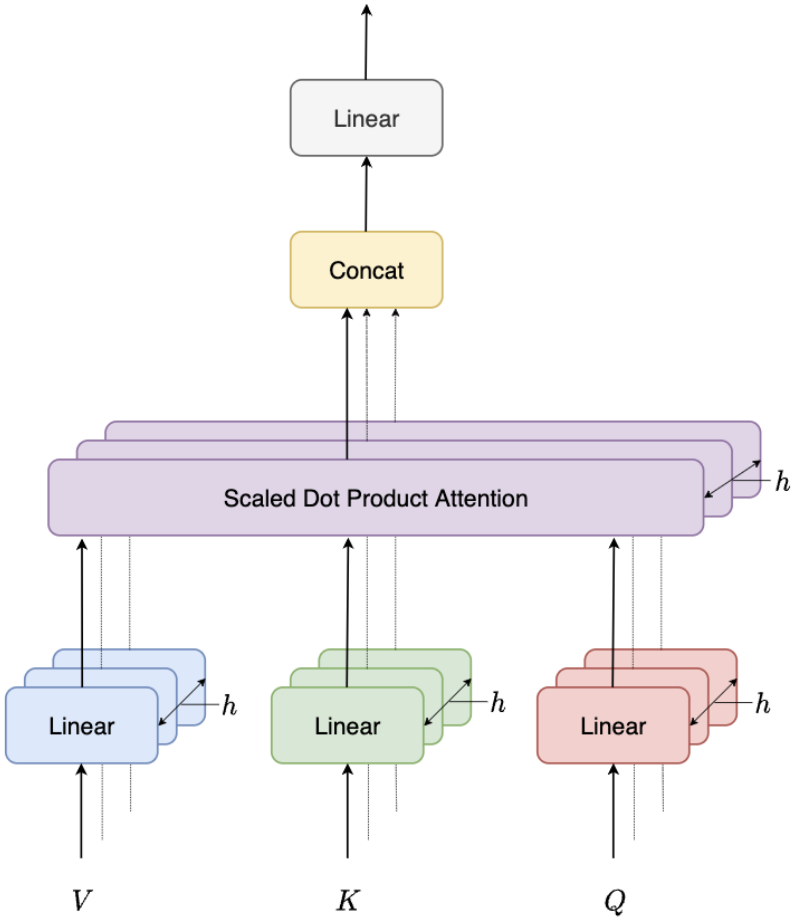


FIGURE 2.21 – MultiHead Attention with h heads

2.6.5 Positional encoding

The attention mechanism is permutation invariant. In order to account for the order of the embedding vectors $X^1, \ldots, X^T \in \mathbb{R}^D$, we use positional encoding vectors.

The positional encoding vectors $p^1, \ldots, p^T \in \mathbb{R}^D$ have the same dimension as the input embeddings $X^1, \ldots, X^T$.

— For each time step $t \in \{1, \ldots, T\}$, there is a unique positional encoding vector.

— The distance between two steps should be consistent across sentences with different lengths.

The method used in the original paper (Vaswani et al., 2017) is a linear transformation $T^{(k)} \in \mathbb{R}^{D \times D}$ such that

$$T^{(k)} p^t = p^{t+k}$$

It is defined as follows : For $d \in \{1, \dots, \frac{D}{2}\}$:

$$w_i = \frac{1}{100000^{\frac{2i}{D}}} \quad \text{and} \quad p^t = \begin{pmatrix} \sin(w_1 t) \\ \cos(w_1 t) \\ \vdots \\ \sin(w_d t) \\ \cos(w_d t) \\ \vdots \\ \sin(w_{\frac{D}{2}} t) \\ \cos(w_{\frac{D}{2}} t) \end{pmatrix}$$

We define for a $d \in \{1, \dots, \frac{D}{2}\}$ and for all $t \in \{1 \dots, T\}$:

$$e_d^t = \begin{pmatrix} \sin w_d t \\ \cos w_d t \end{pmatrix}$$

Therefore,

$$p^t = \begin{pmatrix} e_1^t \\ \vdots \\ e_d^t \\ \vdots \\ e_{\frac{D}{2}}^t \end{pmatrix}$$

By defining :

$$\mathcal{T}^{(k)} = \begin{pmatrix} \Phi_1^{(k)} & & & & \\ & \ddots & & & \\ & & \Phi_d^{(k)} & & \\ & & & \ddots & \\ & & & & \Phi_{\frac{D}{2}}^{(k)} \end{pmatrix} \quad \text{where} \quad \Phi_d^{(k)} = \begin{pmatrix} \cos(w_d k) & \sin(w_d k) \\ -\sin(w_d k) & \cos(w_d k) \end{pmatrix}$$

We get :

$$\mathcal{T}^{(k)}p^t = \begin{pmatrix} \Phi_1^{(k)} & & & & & \\ & \ddots & & & & \\ & & \Phi_d^{(k)} & & & \\ & & & \ddots & & \\ & & & & \Phi_{\frac{D}{2}}^{(k)} \end{pmatrix} \begin{pmatrix} e_1^t \\ \vdots \\ e_d^t \\ \vdots \\ e_{\frac{D}{2}}^t \end{pmatrix}$$

$$= \begin{pmatrix} \Phi_1^{(k)} e_1^t \\ \vdots \\ \Phi_d^{(k)} e_d^t \\ \vdots \\ \Phi_{\frac{D}{2}}^{(k)} e_{\frac{D}{2}}^t \end{pmatrix}$$

We have for all d in $\{1, \ldots, \frac{D}{2}\}$:

$$\begin{aligned} \Phi_d^{(k)} e_d^t &= \begin{pmatrix} \cos(w_d k) & \sin(w_d k) \\ -\sin(w_d k) & \cos(w_d k) \end{pmatrix} \begin{pmatrix} \sin w_d t \\ \cos w_d t \end{pmatrix} \\ &= \begin{pmatrix} \cos(w_d k)\sin w_d t + \sin(w_d k)\cos w_d t \\ -\sin(w_d k)\sin w_d t + \cos(w_d k)\cos w_d t \end{pmatrix} \\ &= \begin{pmatrix} \sin\left(w_d(t+k)\right) \\ \cos\left(w_d(t+k)\right) \end{pmatrix} \\ &= e_d^{t+k} \end{aligned}$$

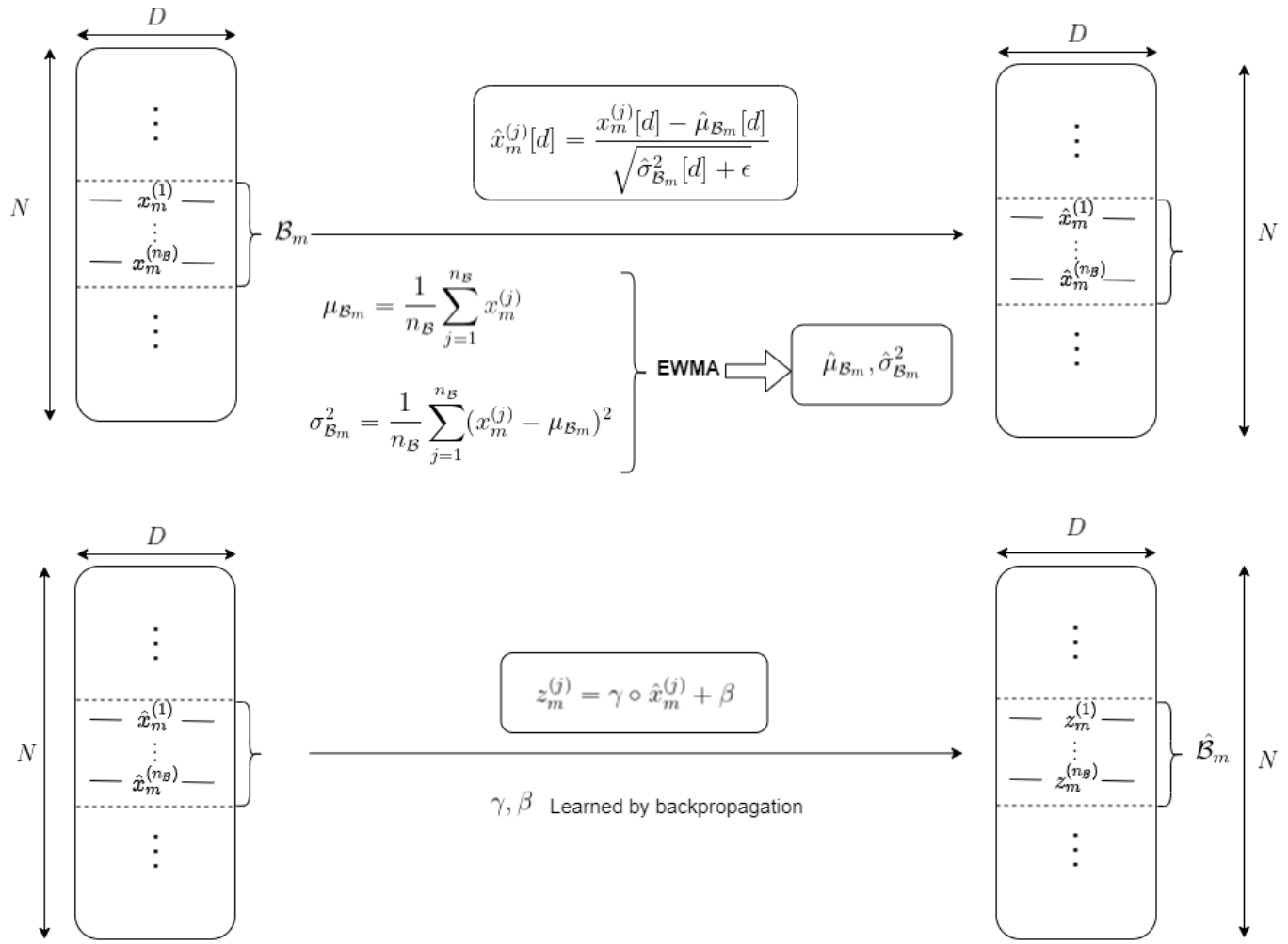
Therefore,

$$\mathcal{T}^{(k)}p^t = \begin{pmatrix} \Phi_1^{(k)} e_1^t \\ \vdots \\ \Phi_d^{(k)} e_d^t \\ \vdots \\ \Phi_{\frac{D}{2}}^{(k)} e_{\frac{D}{2}}^t \end{pmatrix} = \begin{pmatrix} e_1^{t+k} \\ \vdots \\ e_d^{t+k} \\ \vdots \\ e_{\frac{D}{2}}^{t+k} \end{pmatrix} = p^{t+k}$$

2.6.6 The Normalization Process

The Batch Normalization

As explained in (Ioffe & Szegedy, 2015), the Batch Normalization process described in Algorithm 1 considerably helps overcome the vanishing gradients problem by acting like a regularizer.


D
N
$x_m^{(1)}$
$x_m^{(n_\mathcal{B})}$
$\mathcal{B}_m$
$\hat{x}_m^{(j)}[d] = \frac{x_m^{(j)}[d] - \hat{\mu}_{\mathcal{B}_m}[d]}{\sqrt{\hat{\sigma}^2_{\mathcal{B}_m}[d] + \epsilon}}$
$\mu_{\mathcal{B}_m} = \frac{1}{n_\mathcal{B}} \sum_{j=1}^{n_\mathcal{B}} x_m^{(j)}$
$\sigma^2_{\mathcal{B}_m} = \frac{1}{n_\mathcal{B}} \sum_{j=1}^{n_\mathcal{B}} (x_m^{(j)} - \mu_{\mathcal{B}_m})^2$
EWMA
$\hat{\mu}_{\mathcal{B}_m}, \hat{\sigma}^2_{\mathcal{B}_m}$
$\hat{x}_m^{(1)}$
$\hat{x}_m^{(n_\mathcal{B})}$
$z_m^{(j)} = \gamma \circ \hat{x}_m^{(j)} + \beta$
γ, β Learned by backpropagation
$z_m^{(1)}$
$z_m^{(n_\mathcal{B})}$
$\hat{\mathcal{B}}_m$

Algorithm 1 The Batch Normalization

Input:

Batches $\mathcal{B}_1, \ldots, \mathcal{B}_M$ of size $n_{\mathcal{B}} : \forall m \in \{1, \ldots, M\} \quad \mathcal{B}_m = \{x_m^{(1)}, \ldots, x_m^{(n_{\mathcal{B}})}\}$

Output: Batches $\hat{\mathcal{B}}_1, \ldots, \hat{\mathcal{B}}_M$ of size $n_{\mathcal{B}} : \forall m \in \{1, \ldots, M\} \quad \mathcal{B}_m = \{z_m^{(1)}, \ldots, z_m^{(n_{\mathcal{B}})}\}$

1: Set $m_0 = 0, v_0 = 0$

2: **for** $m = 1$ **to** M **do**

3: Calculate the mini-batch mean $\mu_{\mathcal{B}_m}$ and the mini-batch variance $\sigma^2_{\mathcal{B}_m}$ as follows :

$$\mu_{\mathcal{B}_m} = \frac{1}{n_{\mathcal{B}}} \sum_{j=1}^{n_{\mathcal{B}}} x_m^{(j)} \tag{2.6.25}$$

$$\sigma^2_{\mathcal{B}_m} = \frac{1}{n_{\mathcal{B}}} \sum_{j=1}^{n_{\mathcal{B}}} (x_m^{(j)} - \mu_{\mathcal{B}_m}) \circ (x_m^{(j)} - \mu_{\mathcal{B}_m}) \tag{2.6.26}$$

4: Update the mini-batch mean and the mini-batch using EWMA to get $\hat{\mu}_{\mathcal{B}_m}$ and $\hat{\sigma}^2_{\mathcal{B}_m}$ as follows :

$$\hat{\mu}_{\mathcal{B}_m} = \lambda_\mu \circ \hat{\mu}_{\mathcal{B}_{m-1}} + (1 - \lambda_\mu) \circ \mu_{\mathcal{B}_m} \tag{2.6.27}$$

$$\hat{\sigma}^2_{\mathcal{B}_m} = \lambda_\sigma \circ \hat{\sigma}^2_{\mathcal{B}_{m-1}} + (1 - \lambda_\sigma) \circ \sigma^2_{\mathcal{B}_m} \tag{2.6.28}$$

5: Normalize the batch m :

$$\hat{x}_m^{(j)}[d] = \frac{x_m^{(j)}[d] - \hat{\mu}_{\mathcal{B}_m}[d]}{\sqrt{\hat{\sigma}^2_{\mathcal{B}_m}[d] + \epsilon}} \quad (\text{for all } d \in \{1, \ldots, D\} \text{ for all } j \in \{1, \ldots, n_{\mathcal{B}}\}) \tag{2.6.29}$$

with $\epsilon \approx 10^{-5}$ is just a smoothing parameter to avoid dividing by zero.

6: Scale and shift the previous output using two parameters γ and β estimated during the training

$$z_m^{(j)} = \gamma \circ \hat{x}_m^{(j)} + \beta \quad (\text{for all } j \in \{1, \ldots, n_{\mathcal{B}}\}) \tag{2.6.30}$$

Algorithm 1 ensure the distribution of the activations within a layer has zero mean and unit variance across a minibatch.

The Batch Normalization also makes the optimization significantly smoother, as explained in (Santurkar, Tsipras, Ilyas, & Madry, 2018).

The Layer Normalization

The batch normalization process recenters and rescales across the examples within a minibatch. As a result, the batch normalization method can suffer from bad estimates of the mean and the variance parameters when the batch size is too small.

Geoffrey Hinton and his team proposed **Layer Normalization** (Ba, Kiros, & Hinton, 2016). The new algorithm overcomes the cons of Batch Normalization by normalizing the activations across the feature dimension instead of mini-batch directions, which makes is easier to apply for RNNs as well.

2.6.7 The Final Architecture

The Transformer architecture is a sequence to sequence architecture, which relies mainly on the attention mechanism.

The main layers used in the model are the following :

— The Multi-Head Attention layer introduced in 2.6.4.

— The pointwise Feed Forward layer.

— The Normalization Layer 2.6.6

Let us introduce the different steps of both the encoder and the decoder in the transformer.

Consider a sequence $(w_i^1, \ldots, w_i^{T_x})$ of indices representing the sequence of words processed using the word2idx dictionary.

The Encoder Layer

The Encoder, represented in figure 2.23, generates an attention based representation able to focus on a particular piece of information from a large context. It is composed of a stack of $N = 6$ identical layers. Each layer is composed of the following sub-layers :

— The first sub-layer is a Multi-Head self Attention 2.6.4, with a residual connection and a normalization layer .

— The Multi-Head attention can be seen as a way of re-averaging the value vectors in order to create contextual embeddings without introducing non-linearities. The second sub-layer is a fully connected feed-forward layer, with a residual connection and a normalization layer.

In the original paper (Vaswani et al., 2017), all the layers output data of the same dimension $d_{\text{model}} = 512$.

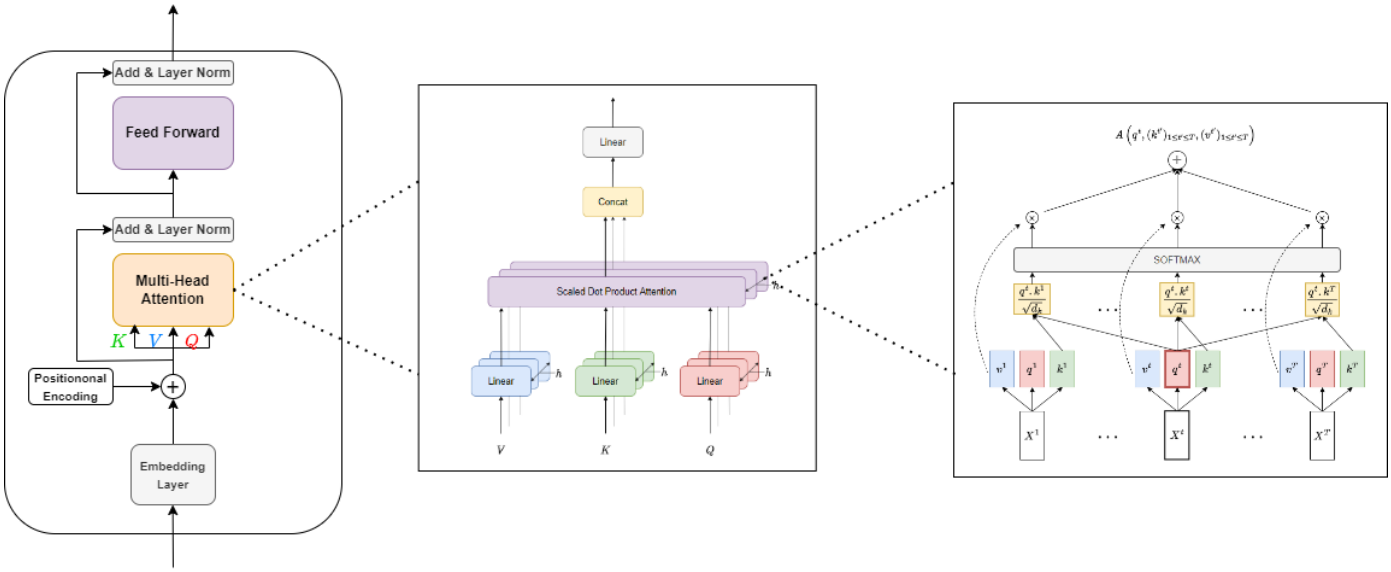


FIGURE 2.23 – The Encoder layer in the Transformer

The Decoder Layer

The decoder, represented in figure 2.24 is responsible of retrieving the information from the encoded representation through the matrices K and V.

It is composed of a stack of $N = 6$ identical layers. Each layer can be divided into the following steps :

— Similarly to the encoder, we combine the multi-head attention layers with a fully connected feed forward layer.

— The first multi-head attention layer is slightly modified to the one introduced in 2.6.4. In order to avoid look-ahead bias we introduce the notion of mask. Consider the following decoder queries, keys and values :

$$Q = \begin{bmatrix} — & q^1 & — \\ \vdots & \vdots & \vdots \\ — & q^T & — \end{bmatrix} \in \mathbb{R}^{T\times d}, \quad K = \begin{bmatrix} — & k^1 & — \\ \vdots & \vdots & \vdots \\ — & k^T & — \end{bmatrix} \in \mathbb{R}^{T\times d}, \quad V = \begin{bmatrix} — & v^1 & — \\ \vdots & \vdots & \vdots \\ — & v^T & — \end{bmatrix} \in \mathbb{R}^{T\times d}$$

For $t \in \{1, \dots, T\}$, we don't want the contextual embedding associated with the query q^t, defined in 2.6.24, to depend on the pairs $(k_{t'}, v_{t'})_{t\leq t'\leq T}$

Hence, we "mask" the contributions $\alpha^{<t,t'>}$ for all $t' > t$ by setting them to zero.

So, we set $e^{<t,t'>}$, defined in 2.6.22, to $-\infty$ for all $t' > t$ as shown in the following table :

	1	2	…	t’	…	T-1	T
1	$e^{<1,1>}$	-∞	…	-∞	…	-∞	-∞
2	$e^{<2,1>}$	$e^{<2,2>}$	…	-∞	…	-∞	-∞
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
t	$e^{<t,1>}$	$e^{<t,2>}$	…	$e^{<t,t'>}$	…	-∞	-∞
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
T-1	$e^{<T-1,1>}$	$e^{<T-1,2>}$	…	$e^{<T-1,t'>}$	…	$e^{<T-1,T-1>}$	-∞
T	$e^{<T,1>}$	$e^{<T,2>}$	…	$e^{<T,t'>}$	…	$e^{<T,T-1>}$	$e^{<T,T>}$

After applying the softmax 2.6.23, we get the masked attention weights :

	1	2	…	t’	…	T-1	T
1	$\alpha^{<1,1>}$	0	…	0	…	0	0
2	$\alpha^{<2,1>}$	$\alpha^{<2,2>}$	…	0	…	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
t	$\alpha^{<t,1>}$	$\alpha^{<t,2>}$	…	$\alpha^{<t,t'>}$	…	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
T-1	$\alpha^{<T-1,1>}$	$\alpha^{<T-1,2>}$	…	$\alpha^{<T-1,t'>}$	…	$\alpha^{<T-1,T-1>}$	0
T	$\alpha^{<T,1>}$	$\alpha^{<T,2>}$	…	$\alpha^{<T,t'>}$	…	$\alpha^{<T,T-1>}$	$\alpha^{<T,T>}$

— Additionally, we add a residual connection and a normalization layer.

— Another multi-head attention layer, where the queries are the output of the first multi-head attention and the keys and values are the encoder outputs.

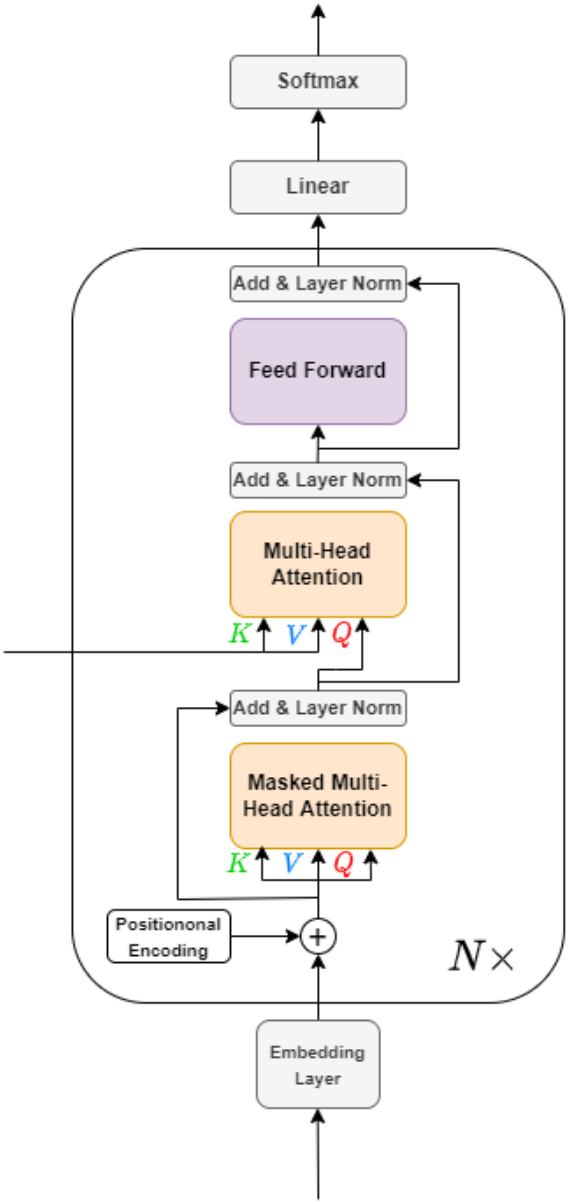


FIGURE 2.24 – The Decoder layer in the Transformer

The Transformer architecture

Finally, by putting it all together, we get the final transformer architecture, represented in 2.25.

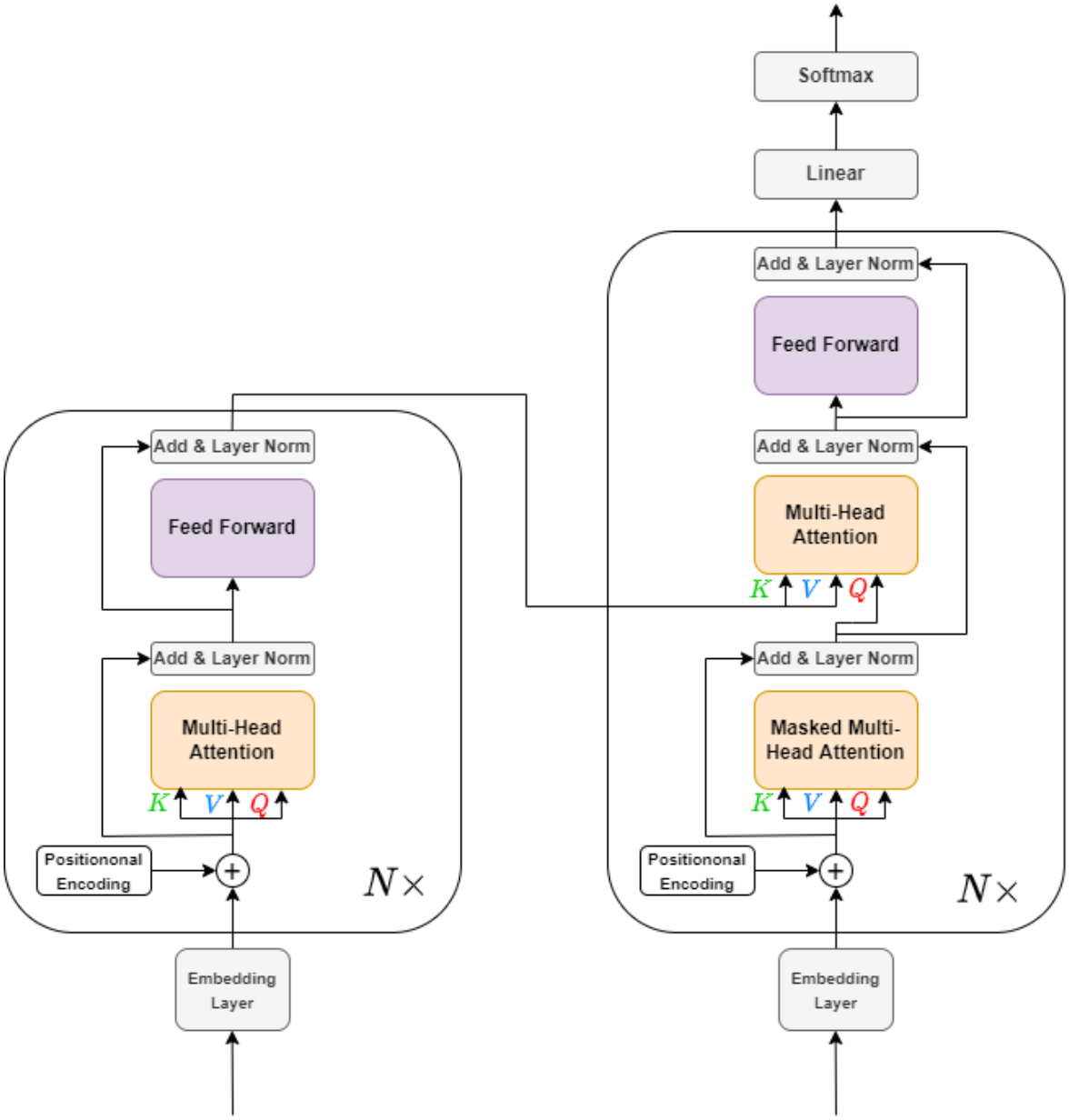


FIGURE 2.25 – The Transformer architecture

2.7 The Optimization algorithm

2.7.1 Position of the problem

In this section, we study the optimization problem, which can be written as follows :

$$\min_{\theta \in \mathbb{R}^d} f(\theta) \quad \text{where} \quad f(\theta) := \mathbb{E}_{s \sim \mathbb{P}}[\mathcal{L}(\theta, s)], \tag{2.7.31}$$

where :

— The function f is called **the objective function**.

— The function $\mathcal{L}$ is the loss function.

— $\mathbb{P}$ is the unknown data distribution on the domain $\mathcal{S}$

— θ is the set of parameters we wish to optimize.

In the following sections, we are going to focus on the Adam optimizer. Section 2.7.2 is a brief introduction to the history of the Adam algorithm. Section 2.7.3 is a description of the algorithm. In section 2.7.4, we analyze the convergence behavior of the algorithm in the nonconvex setting.

2.7.2 Brief history of the Adam optimizer

The Adam algorithm was first introduced in 2015 (Kingma & Ba, 2015). The authors proposed a proof of convergence which was found to have problems. In 2018, (S. J. Reddi et al., 2018) clarified the inconsistency of the previous paper and fixed the proof in the convex setting. In (S. Reddi et al., 2018), the authors conducted the proof for the non convex case under some useful parameter settings.

In the following sections, we provide a detailed version of the proof provided in the original paper (S. Reddi et al., 2018).

2.7.3 Description of the algorithm

The Adam algorithm defined in (S. Reddi et al., 2018) is summarized in Algorithm 2

Algorithm 2 The Adam Optimizer

Input:
Initial parameter value : $\theta_1 \in \mathbb{R}^d$
Learning rates : $\{\eta_t\}_{t=1}^T$
Decay parameters : $0 \leq \beta_1, \beta_2 \leq 1$
Stability parameter : $\epsilon > 0$
Output: ϵ-First Order Stationary Point θ_{T+1}

Set $\boldsymbol{m}_0 = 0, \boldsymbol{v}_0 = 0$
for $t = 1$ **to** T **do**
  Draw a batch $(s_t^i)_{i \in \mathcal{B}_t}$ from $\mathbb{P}$
  Compute $\boldsymbol{g}_t = \frac{1}{|\mathcal{B}_t|} \sum_{s \in \mathcal{B}_t} \nabla \mathcal{L}(\theta_t, s)$
  Update $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\, \boldsymbol{g}_t$
  Update $\boldsymbol{v}_t = \boldsymbol{v}_{t-1} - (1 - \beta_2)\, (\boldsymbol{v}_{t-1} - \boldsymbol{g}_t \circ \boldsymbol{g}_t)$
  Update $\theta_{t+1} = \theta_t - \eta_t \frac{\boldsymbol{m}_t}{\sqrt{\boldsymbol{v}_t + \epsilon}}$

2.7.4 The convergence behavior of the Adam optimizer

Preliminaries

We are fetching for First Order Stationary Points. We would like to prove that under some assumptions on the loss function (not necessarily convex), we can have :

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla f\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq h(T) \quad \text{with} \quad \lim_{T\to+\infty} h(T) = 0$$

Where T is the number of batches

Assumptions

— $(\mathcal{A}_1)$: We assume the loss function $\mathcal{L}$ to be L-smooth, which means that :

$$\forall \theta_1, \theta_2 \in \mathbb{R}^d \; \forall s \in \mathcal{S} \quad \|\nabla\mathcal{L}(\theta_2, s) - \nabla\mathcal{L}(\theta_1, s)\|_2 \leq L\|\theta_2 - \theta_1\|_2 \tag{2.7.32}$$

— $(\mathcal{A}_2)$: We assume the loss function $\mathcal{L}$ to have bounded gradient : i.e,

$$\exists G \in \mathbb{R}_+ \; \forall \theta \in \mathbb{R}^d \; \forall s \in \mathcal{S} \quad \|\nabla\mathcal{L}(\theta, s)\|_2 \leq G \tag{2.7.33}$$

— $(\mathcal{A}_3)$: We assume the variance of the loss function $\mathcal{L}$ to be bounded : i.e,

$$\forall \theta \in \mathbb{R}^d \quad \mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta}; \xi) - \nabla\mathcal{L}(\boldsymbol{\theta})\|_2^2 | \mathcal{F}_t\right] \leq \sigma^2 \tag{2.7.34}$$

where the sigma-algebra $\mathcal{F}_t$ represents *the information known at time t*

The convergence theorem

Theorem 2.7.1 (*Convergence of the Adam Algorithm*)

Let $\eta_t = \eta$ for all $t \in [T]$. Furthermore, assume that ϵ, β_2 and η are chosen such that the following conditions are satisfied :

$$\eta \leq \frac{2G\sqrt{1-\beta_2}}{L} \tag{2.7.35}$$

$$1 - \beta_2 \leq \frac{\epsilon^4}{16G^2(G+\epsilon)^2} \tag{2.7.36}$$

Then, for $(\theta_t)_t$ generated using ADAM (Algorithm 2) , we have the following inequality :

1. *If the batch size b_t is fixed (i.e, $b_t = b_0$ for all t). Then,*

$$\exists c_1, c_2 \in \mathbb{R}_+ \quad \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\theta_t)\|_2^2\right] \leq \frac{c_1}{T} + c_2 \tag{2.7.37}$$

2. *If the batch size $b_t = b_0 T$ for all t. Then,*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_t)\|_2^2\right] = O\left(\frac{1}{T}\right) \tag{2.7.38}$$

3. *If the batch size in linear in time (i.e, $b_t = b_0 t$ for all t). Then,*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_t)\|_2^2\right] = O\left(\frac{\ln(T)}{T}\right) \tag{2.7.39}$$

4. *If the batch size is of the form $b_t = \lceil b_0 t^\gamma \rceil$ for all t (with $0 < \gamma < 1$). Then,*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_t)\|_2^2\right] = O\left(\frac{1}{T^\gamma}\right) \tag{2.7.40}$$

Démonstration : We would like to understand the change in function value between two successive iterations of the algorithm 2. In the whole proof, we will consider $\beta_1 = 0$

1. **Showing that the objective function f is L-smooth**

   For all $\theta_1, \theta_2 \in \mathbb{R}^d$

$$\begin{aligned}
\|\nabla f(\boldsymbol{\theta}_1) - \nabla f(\boldsymbol{\theta}_2)\|_2 &= \|\nabla \mathbb{E}_{s\sim\mathbb{P}}[\mathcal{L}(\boldsymbol{\theta}_1; s)] - \nabla \mathbb{E}_{s\sim\mathbb{P}}[\mathcal{L}(\boldsymbol{\theta}_2; s)]\|_2 \quad \text{(from the definition 4.4.21)} \\
&= \|\mathbb{E}_{s\sim\mathbb{P}}[\nabla\mathcal{L}(\boldsymbol{\theta}_1; s)] - \mathbb{E}_{s\sim\mathbb{P}}[\nabla\mathcal{L}(\boldsymbol{\theta}_2; s)]\|_2 \\
&= \|\mathbb{E}_{s\sim\mathbb{P}}[\nabla\mathcal{L}(\boldsymbol{\theta}_1; s) - \nabla\mathcal{L}(\boldsymbol{\theta}_2; s)]\|_2 \\
&\leq \mathbb{E}_{s\sim\mathbb{P}}[\|\nabla\mathcal{L}(\boldsymbol{\theta}_1; s) - \nabla\mathcal{L}(\boldsymbol{\theta}_2; s)\|_2] \\
&\leq \mathbb{E}_{s\sim\mathbb{P}}[L\|\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1\|_2] \quad \text{(from the assumption 2.7.32)} \\
&= L\|\theta_2 - \theta_1\|_2
\end{aligned}$$

   Therefore, f is L-smooth

2. **Deducing the change in the objective value between two successive iterations**.

   Let us consider $t \in [T]$. As f is L-smooth, we can deduce that :

$$f(\boldsymbol{\theta}_{t+1}) \leq f(\boldsymbol{\theta}_t) + \nabla^\top f(\boldsymbol{\theta}_t)(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) + \frac{L}{2}\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2 \tag{2.7.41}$$

From the update equations in algorithm 2 we have :

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{\mathbf{g}_t}{\left(\sqrt{\mathbf{v}_t} + \epsilon\right)}$$

Which can be expressed component-wise as follows :

$$\forall i \in [d] \quad \boldsymbol{\theta}_{i,t+1} = \boldsymbol{\theta}_{i,t} - \eta_t \frac{\mathbf{g}_{i,t}}{\left(\sqrt{v_{i,t}} + \epsilon\right)} \tag{2.7.42}$$

From 2.7.41 and 2.7.42, we deduce the following inequality :

$$f\left(\boldsymbol{\theta}_{t+1}\right) \leq f\left(\boldsymbol{\theta}_t\right) - \eta_t \sum_{i=1}^{d} \left(\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} \right) + \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \frac{\mathbf{g}_{i,t}^2}{\left(\sqrt{\mathbf{v}_{i,t}} + \epsilon\right)^2}$$

Let us introduce the following notations for each time t' :

b'_t be the size of $\mathcal{B}_{t'}$.

The sigma-algebra $\mathcal{F}_{t'}$ represents *the information known at time t'*.

Consequently :

$$\boxed{\begin{aligned}\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) & \underbrace{-\eta_t \sum_{i=1}^{d} \left(\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} \mid \mathcal{F}_t\right] \right)}_{\text{(a)}} \\ & \underbrace{+ \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\left(\sqrt{\mathbf{v}_{i,t}} + \epsilon\right)^2} \mid \mathcal{F}_t\right]}_{(b)}\end{aligned}} \tag{2.7.43}$$

3. **Bounding the first term (a) in 2.7.43**

We have :

$$\begin{aligned}\mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} \mid \mathcal{F}_t\right] &= \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}} + \epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} + \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] \\ &= \mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} - \frac{\mathbf{g}_{i,t}}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] + \underbrace{\mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]} \\ &= \mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} - \frac{\mathbf{g}_{i,t}}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] + \frac{[\nabla f(\boldsymbol{\theta})]_i}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon}\end{aligned}$$

Which enables us to rewrite (a) defined in 2.7.43 as follows :

$$\begin{aligned}(a) &= -\eta_t \sum_{i=1}^{d} \left(\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \left[\frac{\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} + \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}} + \epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] \right] \right) \\ &= -\eta_t \sum_{i=1}^{d} \frac{\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \underbrace{-\eta_t \sum_{i=1}^{d} \left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}} + \epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]}_{(a_1)}\end{aligned} \tag{2.7.44}$$

Let us bound the term (a_1) in 2.7.44 :

$$
\begin{aligned}
-\eta_t \sum_{i=1}^{d} \left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right] & \\
\leq \left|\eta_t \sum_{i=1}^{d} \left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right]\right| & \\
\leq \eta_t \sum_{i=1}^{d} \left|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i\right| \left| \times \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right]\right| & \\
\leq \eta_t \sum_{i=1}^{d} \left|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i\right| \times \mathbb{E}\left[\underbrace{\left|\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon}\right|}_{(a_2)} \mid \mathcal{F}_t\right] &
\end{aligned}
\tag{2.7.45}
$$

By using the update rule $\mathbf{v}_{i,t} = \beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\,\mathbf{g}_{i,t}^2$ from algorithm 2, we can bound the term (a_2) in 2.7.45 :

$$
\begin{aligned}
(a_2) &= \left|\boldsymbol{g}_{i,t}\right| \left|\frac{1}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{1}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon}\right| \\
&= \frac{\left|\boldsymbol{g}_{i,t}\right|}{\left(\sqrt{\boldsymbol{v}_{i,t}}+\epsilon\right)\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \left|\sqrt{\boldsymbol{v}_{i,t}} - \sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}\right| \\
&= \frac{\left|\boldsymbol{g}_{i,t}\right|}{\left(\sqrt{\boldsymbol{v}_{i,t}}+\epsilon\right)\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \frac{\left|\boldsymbol{v}_{i,t} - \beta_2 \boldsymbol{v}_{i,t-1}\right|}{\sqrt{\boldsymbol{v}_{i,t}} + \sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}} \\
&= \frac{\left|\boldsymbol{g}_{i,t}\right|}{\left(\sqrt{\boldsymbol{v}_{i,t}}+\epsilon\right)\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \frac{(1-\beta_2)\,\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\,\mathbf{g}_{i,t}^2} + \sqrt{\beta_2 \mathbf{v}_{i,t-1}}} \quad \text{(by using the update rule)} \\
&\leq \frac{\left|\boldsymbol{g}_{i,t}\right|}{\left(\sqrt{\boldsymbol{v}_{i,t}}+\epsilon\right)\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \frac{(1-\beta_2)\,\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\,\mathbf{g}_{i,t}^2}} \quad \text{(since } \sqrt{\beta_2 \mathbf{v}_{i,t-1}} \geq 0\text{)} \\
&\leq \frac{\left|\boldsymbol{g}_{i,t}\right|}{\epsilon\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \frac{(1-\beta_2)\,\mathbf{g}_{i,t}^2}{\sqrt{(1-\beta_2)\,\mathbf{g}_{i,t}^2}} \quad \text{(since } \sqrt{v_{i,t}} \geq 0 \text{ and } \beta_2 \mathbf{v}_{i,t-1} \geq 0\text{)} \\
&= \frac{\sqrt{1-\beta_2}\mathbf{g}_{i,t}^2}{\epsilon\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)}
\end{aligned}
\tag{2.7.46}
$$

From 2.7.44, 2.7.45 and 2.7.46, we deduce the following inequality :

$$
(a_1) \leq \eta_t \sum_{i=1}^{d} \left(\left|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i\right| \frac{\sqrt{1-\beta_2}}{\epsilon} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right]\right) \tag{2.7.47}
$$

Therefore, we deduce a bound for (a) from 2.7.44 and 2.7.47 :

$$
(a) \leq -\eta_t \sum_{i=1}^{d} \frac{\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon} + \eta_t \sum_{i=1}^{d} \left(\left|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i\right| \frac{\sqrt{1-\beta_2}}{\epsilon} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right]\right) \tag{2.7.48}
$$

By using the assumption 2.7.33, we can also bound the term $|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i$ for all $i \in [d]$.

Indeed,

$$\begin{aligned}\forall i \in [d] \quad |[\nabla f\left(\boldsymbol{\theta}_t\right)]_i| &\leq \|\nabla f(\boldsymbol{\theta_t})\|_2 \\ &:= \|\mathbb{E}_{s\sim\mathbb{P}}[\mathcal{L}(\theta_t, s)]\|_2 \\ &\leq \mathbb{E}_{s\sim\mathbb{P}}[\|\nabla\mathcal{L}(\boldsymbol{\theta_t}; s)\|] \\ &\leq G \quad \text{(from assumption2.7.33)}\end{aligned}$$

So,

$$\forall i \in [d] \quad |[\nabla f\left(\boldsymbol{\theta}_t\right)]_i| \leq G \tag{2.7.49}$$

From 2.7.48 and 2.7.49 we deduce :

$$\boxed{(a) \leq -\eta_t \sum_{i=1}^{d} \frac{[\nabla f\left(\boldsymbol{\theta}_t\right)]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} + \frac{\eta_t G\sqrt{1-\beta_2}}{\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]} \tag{2.7.50}$$

4. **Bounding the second term (b) in 2.7.43**

By using the update rule $\mathbf{v}_{i,t} = \beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\,\mathbf{g}_{i,t}^2$ from algorithm 2 in the expression (b), we get :

$$\begin{aligned}(b) &:= \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\left(\sqrt{\mathbf{v}_{i,t}} + \epsilon\right)^2} \mid \mathcal{F}_t\right] \\ &= \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}^2}{\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1} + (1-\beta_2)\,\boldsymbol{g}_{i,t}^2} + \epsilon\right)^2} \mid \mathcal{F}_t\right] \\ &\leq \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}^2}{\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon\right)^2} \mid \mathcal{F}_t\right] \quad (\text{since } (1-\beta_2)\,\boldsymbol{g}_{i,t}^2 \geq 0) \\ &\leq \frac{L\eta_t^2}{2\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{g_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] \quad (\text{since} \sqrt{\beta_2 \mathbf{v}_{i,t-1}} \geq 0)\end{aligned}$$

So,

$$\boxed{(b) \leq \frac{L\eta_t^2}{2\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{g_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]} \tag{2.7.51}$$

5. **Combining the upper bounds on (a) and (b) defined in 2.7.43**

By combining the upper bounds 2.7.50 and 2.7.51, we get the following inequality :

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) \underbrace{-\eta_t \sum_{i=1}^{d} \frac{\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon}}_{(c)} + \underbrace{\frac{\eta_t G \sqrt{1-\beta_2}}{\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]}_{(d)} + \underbrace{\frac{L\eta_t^2}{2\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{g_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]}_{(e)} \tag{2.7.52}$$

— Bounding the sum of (d) and (e) defined in 2.7.52 :

We have :

$$\begin{aligned}(d) + (e) &= \left(\frac{\eta_t G \sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t^2}{2\epsilon}\right) \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] \\ &\leq \frac{1}{\epsilon}\left(\frac{\eta_t G \sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t^2}{2\epsilon}\right) \sum_{i=1}^{d} \mathbb{E}\left[\mathbf{g}_{i,t}^2 \mid \mathcal{F}_t\right] \quad (\text{since } \sqrt{\beta_2 \mathbf{v}_{i,t-1}} \geq 0)\end{aligned} \tag{2.7.53}$$

— Bounding the expression (c) defined in 2.7.52 :

To that end, we first need to prove that $\forall i \in [d]\ \forall t' \in [T]\ v_{i,t'} \leq G^2$. We can do it by induction on t'.

— It's true for $t' = 0$

— Let's consider $t' \in [T]$ such that $\forall i \in [d]\ v_{i,t'-1} \leq G^2$. We have :

$$\begin{aligned}\forall i \in [d]\ v_{i,t'} &= \beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\, \mathbf{g}_{i,t}^2 \\ &\leq \beta_2 G^2 + (1-\beta_2)\, \mathbf{g}_{i,t}^2 \quad (\text{by induction hypothesis}) \\ &\leq \beta_2 G^2 + (1-\beta_2)\, \|\mathbf{g}_t\|_2^2 \\ &= \beta_2 G^2 + (1-\beta_2) \left|\left| \frac{1}{b_t} \sum_{s \in \mathcal{B}_t} \nabla \mathcal{L}(\theta_t, s) \right|\right|_2^2 \quad (\text{by definition of } g_t) \\ &\leq \beta_2 G^2 + (1-\beta_2)\, \frac{1}{b_t^2} \sum_{s \in \mathcal{B}_t} ||\nabla \mathcal{L}(\theta_t, s)||_2^2 \\ &\leq \beta_2 G^2 + (1-\beta_2)\, \frac{1}{b_t^2} \sum_{s \in \mathcal{B}_t} G^2 \quad (\text{by assumption 2.7.33}) \\ &= \beta_2 G^2 + (1-\beta_2)\, \frac{1}{b_t^2} b_t G^2 \\ &= \beta_2 G^2 + \frac{(1-\beta_2)}{b_t} G^2 \\ &\leq \beta_2 G^2 + (1-\beta_2)\, G^2 \quad (\text{since} b_t \geq 1) \\ &= G^2\end{aligned}$$

We conclude by induction that,

$$\forall i \in [d]\ \forall t' \in [T]\ v_{i,t'} \leq G^2 \tag{2.7.54}$$

Consequently,

$$\begin{aligned}(c) &:= -\eta_t \sum_{i=1}^{d} \frac{[\nabla f(\boldsymbol{\theta}_t)]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \\ &\leq -\frac{\eta_t}{\sqrt{\beta_2} G + \epsilon} \sum_{i=1}^{d} [\nabla f(\boldsymbol{\theta}_t)]_i^2 \quad \text{(by using 2.7.54)} \\ &= -\frac{\eta_t}{\sqrt{\beta_2} G + \epsilon} \|\nabla f(\boldsymbol{\theta}_t)\|_2^2 \end{aligned} \tag{2.7.55}$$

— Combining the results :

By using the inequalities 2.7.53 and 2.7.55, the upper bound in 2.7.52 becomes :

$$\boxed{\begin{aligned}\mathbb{E}[f(\boldsymbol{\theta}_{t+1}) \mid \mathcal{F}_t] \leq f(\boldsymbol{\theta}_t) &- \frac{\eta_t}{\sqrt{\beta_2} G + \epsilon} \|\nabla f(\boldsymbol{\theta}_t)\|_2^2 \\ &+ \frac{1}{\epsilon} \left(\frac{\eta_t G \sqrt{1-\beta_2}}{\epsilon} + \frac{L \eta_t^2}{2\epsilon} \right) \underbrace{\mathbb{E}\left[\|\mathbf{g}_t\|_2^2 \mid \mathcal{F}_t \right]}_{(f)}\end{aligned}} \tag{2.7.56}$$

6. **Bounding the last term (f)**

Let us introduce the following notations :

$$\xi_t := \frac{1}{b_t} \sum_{s \in \mathcal{B}_t} (\nabla \mathcal{L}(\theta_t, s) - \nabla f(\theta_t)) \tag{2.7.57}$$

$$\forall s \in \mathcal{B}_t \quad Y_s := \nabla \mathcal{L}(\theta_t, s) - \nabla f(\theta_t) \tag{2.7.58}$$

$$Y := \sum_{s \in \mathcal{B}_t} Y_s \tag{2.7.59}$$

Then,

$$\xi_t := \frac{1}{b_t} Y \tag{2.7.60}$$

Consequently,

$$
\begin{aligned}
(f) &:= \mathbb{E}\left[\|\mathbf{g}_t\|_2^2 \mid \mathcal{F}_t\right] \\
&= \mathbb{E}\left[\left\|\frac{1}{b_t}\sum_{s\in\mathcal{B}_t}\nabla\mathcal{L}(\theta_t,s)\right\|_2^2 \mid \mathcal{F}_t\right] \quad \text{(by definition)} \\
&= \mathbb{E}\left[\left\|\frac{1}{b_t}\sum_{s\in\mathcal{B}_t}\left(\nabla\mathcal{L}(\theta_t,s)-\nabla f(\theta_t)+\nabla f(\theta_t)\right)\right\|_2^2 \mid \mathcal{F}_t\right] \\
&= \mathbb{E}\left[\left\|\left(\frac{1}{b_t}\sum_{s\in\mathcal{B}_t}\left(\nabla\mathcal{L}(\theta_t,s)-\nabla f(\theta_t)\right)\right)+\nabla f(\theta_t)\right\|_2^2 \mid \mathcal{F}_t\right] \\
&= \mathbb{E}\left[\|\xi_t+\nabla f(\theta_t)\|_2^2 \mid \mathcal{F}_t\right] \quad \text{(by definition 2.7.57)} \\
&= \mathbb{E}\left[\left(\xi_t+\nabla f(\theta_t)\right)^\top\left(\xi_t+\nabla f(\theta_t)\right) \mid \mathcal{F}_t\right] \\
&= \mathbb{E}\left[\|\xi_t\|_2^2 \mid \mathcal{F}_t\right] + \underbrace{\mathbb{E}\left[\xi_t \mid \mathcal{F}_t\right]^\top}_{=0}\nabla f(\theta_t) + \nabla f(\theta_t)^\top\underbrace{\mathbb{E}\left[\xi_t \mid \mathcal{F}_t\right]}_{=0} + \|\nabla f(\theta_t)\|_2^2 \\
&= \frac{1}{b_t^2}\mathbb{E}\left[\|Y\|_2^2 \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \quad \text{(using 2.7.60)} \\
&= \frac{1}{b_t^2}\mathbb{E}\left[\left\|\left(\sum_{s\in\mathcal{B}_t}Y_s\right)^\top\left(\sum_{s'\in\mathcal{B}_t}Y_{s'}\right)\right\|_2^2 \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \quad \text{(using 2.7.59)} \\
&= \frac{1}{b_t^2}\sum_{s,s'\in\mathcal{B}_t}\mathbb{E}\left[Y_s^\top Y_{s'} \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \\
&= \frac{1}{b_t^2}\sum_{\substack{s,s'\in\mathcal{B}_t\\ s\neq s'}}\mathbb{E}\left[Y_s^\top Y_{s'} \mid \mathcal{F}_t\right] + \frac{1}{b_t^2}\sum_{\substack{s,s'\in\mathcal{B}_t\\ s=s'}}\mathbb{E}\left[Y_s^\top Y_{s'} \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \\
&= \frac{1}{b_t^2}\sum_{\substack{s,s'\in\mathcal{B}_t\\ s\neq s'}}\mathbb{E}\left[Y_s \mid \mathcal{F}_t\right]^\top\underbrace{\mathbb{E}\left[Y_{s'} \mid \mathcal{F}_t\right]}_{=0} + \frac{1}{b_t^2}\sum_{s\in\mathcal{B}_t}\mathbb{E}\left[\|Y_s\|_2^2 \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \quad (\text{since} Y_s Y_{s'} \mid \mathcal{F}_t) \\
&= \frac{1}{b_t^2}\sum_{s\in\mathcal{B}_t}\mathbb{E}\left[\|\nabla\mathcal{L}(\theta_t,s)-\nabla f(\theta_t)\|_2^2 \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \quad \text{(by definition 2.7.58)} \\
&\leq \frac{1}{b_t^2}\sum_{s\in\mathcal{B}_t}\sigma^2 + \|\nabla f(\theta_t)\|_2^2 \quad \text{(using the assumption 2.7.34)} \\
&\leq \frac{\sigma^2}{b_t} + \|\nabla f(\theta_t)\|_2^2
\end{aligned}
$$

We conclude the following bound on the (f) term defined in 2.7.56 :

$$(f) \leq \frac{\sigma^2}{b_t} + \|\nabla f(\theta_t)\|_2^2 \tag{2.7.61}$$

By using the bound 2.7.61, the inequality 2.7.56 becomes :

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) - \frac{\eta_t}{\sqrt{\beta_2}G+\epsilon}\left\|\nabla f\left(\boldsymbol{\theta}_t\right)\right\|_2^2 + \frac{1}{\epsilon}\left(\frac{\eta_t G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t^2}{2\epsilon}\right)\left(\frac{\sigma^2}{b_t} + \|\nabla f(\theta_t)\|_2^2\right)$$

Which results in the following inequality :

$$\boxed{\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) + \left\|\nabla f\left(\boldsymbol{\theta}_t\right)\right\|_2^2 \underbrace{\left(\frac{-\eta_t}{\sqrt{\beta_2}G + \epsilon} + \frac{1}{\epsilon}\left(\frac{\eta_t G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t^2}{2\epsilon}\right)\right)}_{(i)} + \underbrace{\frac{\eta_t \sigma^2}{\epsilon b_t}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t}{2\epsilon}\right)}_{(ii)}} \tag{2.7.62}$$

7. **Incorporating the assumptions on the hyperparameters**

The final step of the proof is to use the conditions on the hyperparameters to bound (i) and (ii).

— Using the choice of η

Based on the condition 2.7.36, the learning rate is chosen to be fixed such that :

$$\eta \leq \frac{2G\sqrt{1-\beta_2}}{L}$$

Therefore,

$$\frac{L\eta}{2\epsilon} \leq \frac{G\sqrt{1-\beta_2}}{\epsilon} \tag{2.7.63}$$

We can then bound the first term (i) as follows :

$$\begin{aligned}
(i) &:= -\eta\left(\frac{1}{\sqrt{\beta_2}G + \epsilon} - \frac{1}{\epsilon}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t}{2\epsilon}\right)\right) \\
&\leq -\eta\left(\frac{1}{\sqrt{\beta_2}G + \epsilon} - \frac{1}{\epsilon}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{G\sqrt{1-\beta_2}}{\epsilon}\right)\right) \quad \text{(using 2.7.63)} \\
&= -\eta\left(\frac{1}{\sqrt{\beta_2}G + \epsilon} - \underbrace{\frac{2G\sqrt{1-\beta_2}}{\epsilon^2}}_{(iii)}\right)
\end{aligned} \tag{2.7.64}$$

We can use the inequality 2.7.63 to bound the second term (ii) as follows :

$$\begin{aligned}
(ii) &:= \frac{\eta_t \sigma^2}{\epsilon b_t}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t}{2\epsilon}\right) \\
&\leq \frac{\eta_t \sigma^2}{\epsilon b_t}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{G\sqrt{1-\beta_2}}{\epsilon}\right) \quad \text{(using 2.7.63)} \\
&= \frac{2\eta\sigma^2 G\sqrt{1-\beta_2}}{\epsilon^2 b_t}
\end{aligned} \tag{2.7.65}$$

— Using the choice of β_2 to bound (iii)

By using condition 2.7.36, we can bound (iii) :

$$\begin{aligned}\frac{2G\sqrt{1-\beta_2}}{\epsilon^2} &= \sqrt{\frac{4G^2(1-\beta_2)}{\epsilon^4}} \\ &\leq \sqrt{\frac{4G^2}{\epsilon^4}\frac{\epsilon^4}{16G^2(G+\epsilon)^2}} \quad \text{(using 2.7.36)} \\ &= \frac{1}{2}\frac{1}{G+\epsilon} \\ &\leq \frac{1}{2}\frac{1}{\sqrt{\beta_2}G+\epsilon} \quad \text{(since } \beta_2 \leq 1\text{)} \end{aligned} \tag{2.7.66}$$

— Deducing a new bound for (i)

From 2.7.66, we conclude that :

$$\frac{1}{\sqrt{\beta_2}G+\epsilon} - \frac{2G\sqrt{1-\beta_2}}{\epsilon^2} \geq \frac{1}{2}\frac{1}{\sqrt{\beta_2}G+\epsilon}$$

The inequality 2.7.64 becomes

$$(i) \leq -\frac{\eta}{2(\sqrt{\beta_2}G+\epsilon)} \tag{2.7.67}$$

Finally, by using 2.7.67 and 2.7.65, we get the following update to the inequality 2.7.62 :

$$\boxed{\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) - \frac{\eta}{2(\sqrt{\beta_2}G+\epsilon)}\left\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\right\|_2^2 + \frac{2\eta\sigma^2G\sqrt{1-\beta_2}}{\epsilon^2 b_t}} \tag{2.7.68}$$

8. **Concluding according to the batch size**

Notations :

Let us define the following constants :

$$\begin{aligned}\Delta &= \frac{\eta}{2(\sqrt{\beta_2}G+\epsilon)} \\ \alpha &= \frac{2\eta\sigma^2G\sqrt{1-\beta_2}}{\epsilon^2}\end{aligned}$$

The inequality 2.7.68 can then be written as follows :

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) - \Delta\left\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\right\|_2^2 + \frac{\alpha}{b_t} \tag{2.7.69}$$

By taking the expected value of the inequality 2.7.69,

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right)\right] \leq \mathbb{E}\left[f\left(\boldsymbol{\theta}_t\right)\right] - \Delta\mathbb{E}\left[\left\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\right\|_2^2\right] + \frac{\alpha}{b_t}$$

Which can be rearranged as follows :

$$\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \frac{\mathbb{E}\left[f\left(\boldsymbol{\theta}_t\right)\right] - \mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right)\right]}{\Delta} + \frac{\alpha}{\Delta b_t} \tag{2.7.70}$$

By summing 2.7.70 for all $t \in [T]$, we get :

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] &\leq \frac{1}{T\Delta}\sum_{t=1}^{T}\left(\mathbb{E}\left[f\left(\boldsymbol{\theta}_t\right)\right] - \mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right)\right]\right) + \frac{\alpha}{T\Delta}\sum_{t=1}^{T}\frac{1}{b_t} \\
&= \frac{1}{T\Delta}\left(f\left(\boldsymbol{\theta}_1\right) - \mathbb{E}\left[f\left(\boldsymbol{\theta}_{T+1}\right)\right]\right) + \frac{\alpha}{T\Delta}\sum_{t=1}^{T}\frac{1}{b_t} \quad \text{(using telescoping sum)} \\
&\leq \frac{1}{T\Delta}\left(f\left(\boldsymbol{\theta}_1\right) - f\left(\boldsymbol{\theta}^*\right)\right) + \frac{\alpha}{T\Delta}\sum_{t=1}^{T}\frac{1}{b_t} \quad \left(\text{where } \theta^* := \arg\min_{\theta} f(\theta)\right)
\end{aligned}$$

We conclude that :

$$\boxed{\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \frac{f\left(\boldsymbol{\theta}_1\right) - f\left(\boldsymbol{\theta}^*\right)}{T\Delta} + \frac{\alpha}{T\Delta}\sum_{t=1}^{T}\frac{1}{b_t}} \tag{2.7.71}$$

— If the batch size is fixed : $b_t = b_0$ for all t :

Then, the inequality 2.7.71 becomes :

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \frac{f\left(\boldsymbol{\theta}_1\right) - f\left(\boldsymbol{\theta}^*\right)}{T\Delta} + \frac{\alpha}{\Delta b_0}$$

Let us denote $c_1 = \frac{f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}^*)}{\Delta}$ and $c_2 = \frac{\alpha}{\Delta b_0}$, we conclude the first part 2.7.37 of the theorem :

$$\boxed{\exists c_1, c_2 \in \mathbb{R}_+ \quad \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \frac{c_1}{T} + c_2}$$

— If the batch size $b_t = b_0 T$ for all t :

Then, the inequality 2.7.71 becomes :

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] &\leq \frac{c_1}{T} + \frac{c_2}{T}\sum_{t=1}^{T}\frac{1}{T} \\
&= \frac{c_1 + c_2}{T}
\end{aligned}$$

We conclude the part 2.7.38 of the theorem, i.e :

$$\boxed{\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] = O\left(\frac{1}{T}\right)}$$

— If the batch size in linear in time (i.e, $b_t = b_0 t$ for all t) :

Then, the inequality 2.7.71 becomes :

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \underbrace{\frac{c_1}{T}+\frac{c_2}{T}\sum_{t=1}^{T}\frac{1}{t}}_{(\mathcal{E}_1)} \tag{2.7.72}$$

We would like to find an equivalent to $(\mathcal{E}_1)$.

By applying the mean value theorem to the function $\Phi : t \mapsto \ln(t)$ between t and $t+1$ (for a fixed $t \in [T]$), there exists $c_t \underset{t\to+\infty}{\sim} t$ such that :

$$\Phi(t+1)-\Phi(t) = \frac{1}{c_t} \sim \frac{1}{t}$$

As, $\sum(\frac{1}{t})_t$ diverges, we conclude that :

$$\sum_{t=1}^{T}\left(\Phi(t+1)-\Phi(t)\right) \underset{t\to+\infty}{\sim} \sum_{t=1}^{T}\frac{1}{t}$$

By telescoping sum, it implies :

$$\ln(T) \underset{T\to+\infty}{\sim} \sum_{t=1}^{T}\frac{1}{t}$$

Which gives, by deviding by T :

$$\frac{1}{T}\sum_{t=1}^{T}\frac{1}{t} \underset{T\to+\infty}{\sim} \frac{\ln(T)}{T}$$

On the other hand,

$$\frac{1}{T} = o\left(\frac{\ln(T)}{T}\right)$$

Which gives an equivalent to the bound $(\mathcal{E}_1)$ in 2.7.72 :

$$(\mathcal{E}_1) \underset{T\to+\infty}{\sim} c_2\frac{\ln(T)}{T} \tag{2.7.73}$$

From 2.7.71 and 2.7.73 we conclude the part 2.7.39 of the theorem :

$$\boxed{\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla f\left(\boldsymbol{\theta}_t\right)\|_2^2\right] = O\left(\frac{\ln(T)}{T}\right)}$$

— If the batch size is of the form $b_t = \lceil b_0 t^\gamma \rceil$ for all t (with $0 < \gamma < 1$) :

Then, the inequality 2.7.71 becomes :

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \underbrace{\frac{c_1}{T}+\frac{c_2}{T}\sum_{t=1}^{T}\frac{1}{t^\gamma}}_{(\mathcal{E}_2)} \tag{2.7.74}$$

We would like to find an equivalent to $(\mathcal{E}_2)$:

By applying the mean value theorem to the function $\Psi : t \mapsto \frac{1}{1-\gamma} t^{1-\gamma}$ between $t-1$ and t, there exists $c_t \sim t$ such that :

$$\Psi(t) - \Psi(t-1) = \frac{1}{c_t^{\gamma}} \underset{t \to +\infty}{\sim} \frac{1}{t^{\gamma}}$$

And the Riemann series $\sum_t \frac{1}{t^{\gamma}}$ diverges (since $\gamma < 1$), so we have :

$$\sum_{t=1}^{T} (\Psi(t) - \Psi(t-1)) \underset{t \to +\infty}{\sim} \sum_{t=1}^{T} \frac{1}{t^{\gamma}}$$

Hence, by telescoping sum :

$$\frac{T^{1-\gamma}}{1-\gamma} \underset{T \to +\infty}{\sim} \sum_{t=1}^{T} \frac{1}{t^{\gamma}}$$

Consequently :

$$\frac{c_2}{T} \sum_{t=1}^{T} \frac{1}{t^{\gamma}} \underset{T \to +\infty}{\sim} \frac{c_2}{(1-\gamma) T^{\gamma}}$$

And since :

$$\frac{1}{T} = o\left(\frac{1}{T^{\gamma}}\right)$$

We conclude the following equivalent to the bound $(\mathcal{E}_2)$:

$$(\mathcal{E}_2) \underset{T \to +\infty}{\sim} \frac{c_2}{(1-\gamma) T^{\gamma}} \tag{2.7.75}$$

From 2.7.71 and 2.7.75 we conclude the second part 2.7.39 of the theorem :

$$\boxed{\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\left\| \nabla f\left(\boldsymbol{\theta}_t\right) \right\|_2^2 \right] = O\left(\frac{1}{T^{\gamma}}\right)}$$

■

3 | Forecasting Market Turbulence regimes using Latent Spectral Representation

Asset allocation is one of the most challenging problems of Modern Finance. For many decades, the financial research in this area relied on Markowitz theory and its refinements to build investment portfolios. We propose a robust asset allocation method based on an interpretable low-dimensional representation framework of financial time series combining signal processing, deep neural networks along with bayesian statistics. This representation framework exhibits particularly good clustering properties, which enables us to define two market regimes : The High and the Low Turbulence regimes. By modeling their dynamics, we are able to forecast the market regime over the next period of time. Hence, we enrich the paradigm of asset management by reducing investable periods for any risky asset to their low turbulence periods and alternatively invest in safe assets (typically cash or treasuries) during high turbulence periods. By doing so, we not only enable investors to enjoy smoother "investment journeys", but we also aim at generating reasonable returns by considerably reducing the volatility and the drawdowns along the way.

3.1 Introduction

The Low Turbulence approach, illustrated in figure 3.1, aims to capture the regimes associated with an index and predict the probability of being in each of them in the future. Typically, these regimes are well characterized and remain stable over time.

To do so, the first step (step (a)) consists in extracting the frequency content of the signal. As financial time series are highly non stationary signals, a frequency analysis of such signals has to be localized in time. Inspired by quantum mechanics, the physicist Gabor proposed in 1946 a way of decomposing signals over elementary functions that have a narrow localization in time and frequency, called **time-frequency atoms**. (Mallat, 1999) provides a detailed description of it. Hence, by slicing the series of daily returns into overlapped frames and applying a discrete version of the Gabor Transform, each window is associated with a spectral vector. Each dimension of the spectral vector represents a range of frequencies.

The second step (step (b)) consists in modeling the distribution of the spectral vectors using a conditional variational auto-encoder (CVAE).

Once the CVAE is trained, we use the encoder to map the spectral vectors of the MSCI sectors sub-indices to their low-dimensional representation called the **latent spectral vectors**.

Our main finding is that by clustering the latent spectral vectors into two clusters, the resulting clusters tend to be well separated, which suggests the existence of two regimes called the High and Low Turbulence regimes. The low turbulence regime is characterized by lower volatility and better returns in

comparison with the high turbulence regime. Moreover, these regimes tend to be very stable over time.

Hence, for each sector sub-index, step (c) consists in using a graphical model to learn the latent regimes and their dynamics. To that end, We have used the Hidden Markov Model, introduced in (Rabiner & Juang, 1986). Once the HMM model is trained, we can infer the probability of being in a low turbulence regime over the next period of time conditioned on the past latent spectral vectors.

Aggregating the probabilities associated with the different sectors sub-indices enables us to determine the final allocation among the geographical index, cash and treasuries.

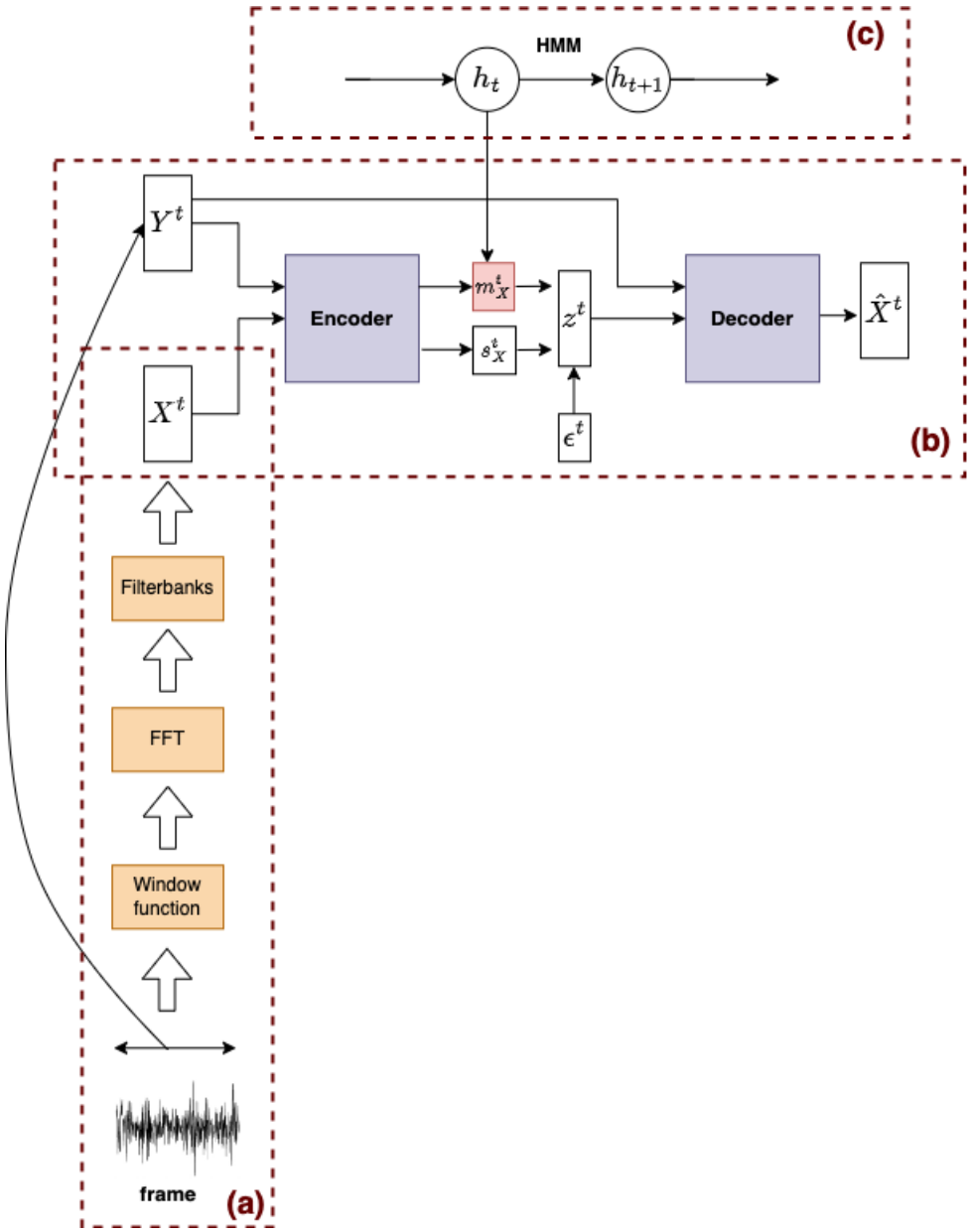


FIGURE 3.1 – The Low Turbulence Model

3.2 Related Work

One of the most revolutionary ideas of Modern Finance in asset allocation was proposed by Markowitz in 1952 ((Markowitz et al., 1952)). The problem was written as a convex optimization problem aiming at estimating an *efficient frontier* of portfolios that maximizes the expected return under a given level of risk. Black and Litterman ((Black & Litterman, 1992)) attempted to regularize the solution by adding some additional information regarding the mean or variance. Ledoit and Wolf ((Ledoit & Wolf, 2004) suggested improving the numerical stability of the covariance matrix's inverse.

The research field has evolved over the past 50 years from traditional asset allocation methods involving the estimation of a covariance matrix to the extensive use of artificial intelligence methods in a supervised or unsupervised manner. Machine learning algorithms have indeed proven successful to model complex data by learning useful data representations.

We consider here the task of extracting meaningful representations of time series by combining time frequency analysis and deep generative models.

Time-Frequency analysis was introduced in order to develop a mathematical representation of musical notation. Indeed, when we listen to music, we can "hear" the time variation of the frequencies. Physically speaking, it does not make sense to consider a signal as a superposition of functions that lack any time localization properties. Windowed Fourier transforms and Wavelet transforms are two classes of local time-frequency decomposition. Over the last two decades, time-frequency analysis has been studied extensively on a theoretical level (Cohen, 1989), (Bayram & Baraniuk, 1996). Ville introduced the Wigner distribution as a time frequency density, and Gabor introduced the decomposition of time frequencies into "time-frequency atoms". Appendix A.1 provides a brief description of the time frequency analysis of the Fourier atoms, as described in (Mallat, 1999).

The intuition behind generative models can be summarized by the famous quote from Richard Feynmann : "What I cannot create, I do not understand". As a matter of fact, these models view the world from the perspective of probability. As a result, any observed data can be viewed as a finite set of samples from an underlying distribution. A generative model's primary objective is to approximate the data distribution given access to the data. The most well-known application of such models consists in generating images, texts, and audio. A generative adversarial network (GAN) can achieve better image fidelity than a variational autoencoder, whereas a variational autoencoder is more stable and can be used to estimate the probability distribution itself. As of today, Variational autoencoders (VAEs) stand among the most powerful class of deep generative models. This probabilistic model can infer a rich hidden structure in the underlying data, which enables them to generate a wide range of complicated

data, including handwritten digits ((Kingma & Welling, 2013) (Salimans, Kingma, & Welling, 2015)), faces ((Rezende, Mohamed, & Wierstra, 2014), (Kulkarni, Whitney, Kohli, & Tenenbaum, 2015)), house numbers ((Kingma, Mohamed, Jimenez Rezende, & Welling, 2014), (Gregor, Danihelka, Graves, Rezende, & Wierstra, 2015)), CIFAR images (Gregor et al., 2015), physical models of scenes (Kulkarni et al., 2015), segmentation (Sohn, Lee, & Yan, 2015), predicting the future from static images (Walker, Doersch, Gupta, & Hebert, 2016), and synthetic data generation in the time-series domain (Desai, Freeman, Wang, & Beaver, 2021).

VAEs combine bayesian variational inference with encoding decoding neural networks in order to learn the probability distribution of the input data. The encoding decoding paradigm is used in several areas of Machine Learning. For instance, in Natural Language Processing, universal encoding vectors can be extracted using a neural network aiming at predicting some context vectors. It results in a mapping from the high dimensional space of words into a low dimensional feature space with interesting geometrical properties ((Mikolov, Sutskever, Chen, Corrado, & Dean, 2013)). More recently, universal embeddings can be extracted from encoding-decoding architectures trained for machine translation((McCann, Bradbury, Xiong, & Socher, 2017)). Likewise, the speech community has used the bottleneck features trained on phoneme predictions (Veselỳ, Karafiát, Grézl, Janda, & Egorova, 2012).

The Low Turbulence approach uses Hidden Markov Models to learn the dynamics of latent spectral vectors derived from the generative model.

Hidden Markov Models (HMMs) are powerful graphical models used to describe sequential data. They were first introduced in a series of statistical papers by Leonard E. Baum (Baum & Petrie, 1966) and other authors in the second half of the 1960s. Besides their rich mathematical structure, HMMs work very well in a wide range of real world applications. Due to the successful results in Speech Recognition (Rabiner & Juang, 1986), the model has become increasingly popular and has been applied to several other areas such as natural language modeling (Manning & Schutze, 1999). There are also many applications in finance : (S. Kim et al., 1998) used HMMs for the analysis of stochastic volatility, in comparison with the traditional GARCH model. (Nystrup et al., 2017) used the model to infer the hidden states associated with the daily returns in financial markets. Other applications include handwriting recognition (Nag et al., 1986) and bioinformatics (Krogh et al., 1994), (Durbin et al., 1998), (Baldi & Brunak, 2001), (Manogaran et al., 2018), (Testa et al., 2015).

To the best of our knowledge, no previous work has combined time frequency analysis with the aforementioned graphical models to detect regimes in time series.

3.3 Methodology

3.3.1 Extracting the spectral vectors from the asset returns

The different steps involved in the extraction of the spectral vectors from the daily returns are detailed in figure 3.2.

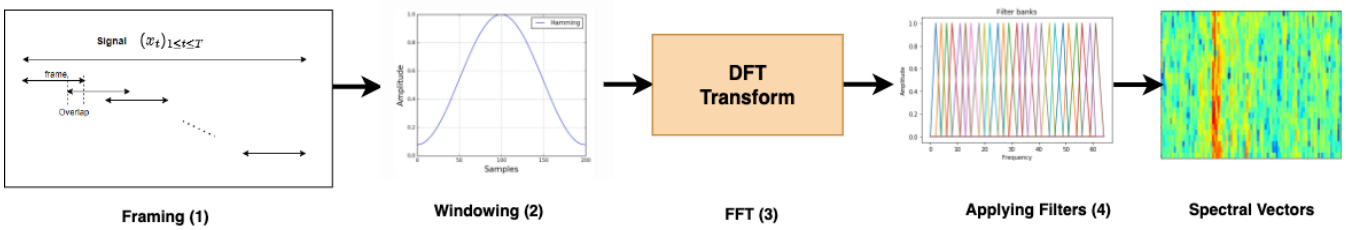


FIGURE 3.2 – Different steps involved in computing the spectral vectors

Let us dive into the details of each step.

1. **Step 1 : Framing**

   Since the frequency structure of a signal changes over time, especially for highly non stationary signals like daily returns, it does not make sense to perform the Fourier transform across the entire signal.

   Therefore, we need to split the signal composed of daily returns $(x_t)_{1\leq t\leq T}$ into short-time frames of size N, with an overlap of size $m < N$. Therefore, we get $L = \lfloor \frac{T-N}{N-m} \rfloor + 1$ frames of length N.

   We denote $f_p = [f_p^0, \ldots, f_p^{N-1}]$ the p-th frame.

2. **Step 2 : Windowing**

   Once the signal has been sliced into frames, we apply a window function such as the Hamming window to each frame. The windowed Fourier approach replaces the Fourier transform's sinusoidal wave by the product of a sinusoid and a window well localized in time and frequency called the Fourier atoms.

3. **Step 3 : The Discrete Fourier Transform**

   At this stage, we extract the spectral information for each frame. We would like to know how much energy the signal contains per frequency band. The adequate tool for extracting spectral information for discrete frequency bands for a discrete-time signal is the Discrete Fourier Transform or DFT.

   Let $s_p = [s_p^0, \ldots, s_p^{N-1}]$ be the vector resulting from the pointwise multiplication between the p-

th frame $[f_p^0, \ldots, f_p^{N-1}]$ and $h = [h^0, \ldots, h^{N-1}]$, an N sample long window function. The k-th element of the discrete Fourier transform is given by :

$$\hat{f}_p^k = \sum_{j=0}^{N-1} f_p^j h^j e^{\frac{-2i\pi jk}{N}} = \sum_{j=0}^{N-1} s_p^j e^{\frac{-2i\pi jk}{N}}$$

The DFT is therefore a linear operator that maps the data points in s_p to the frequency domain $\hat{f}_p$.

We can express the DFT using a simple matrix multiplication as follows :

$$\begin{bmatrix} \hat{f}_p^0 \\ \hat{f}_p^1 \\ \vdots \\ \hat{f}_p^{N-2} \\ \hat{f}_p^{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w_N & w_N^2 & \ldots & w_N^{N-1} \\ 1 & w_N^2 & w_N^4 & \ldots & w_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_N^{N-1} & w_N^{2(N-1)} & \ldots & w_N^{(N-1)^2} \end{bmatrix} \begin{bmatrix} s_p^0 \\ s_p^1 \\ \vdots \\ s_p^{N-2} \\ s_p^{N-1} \end{bmatrix} \tag{3.3.1}$$

where $w_N = e^{\frac{-2i\pi}{N}}$

The vector $\hat{f}_p = [\hat{f}_p^0, \hat{f}_p^1, \ldots, \hat{f}_p^{N-1}]$ contains the Fourier coefficients for the input vector s_p, and the DFT matrix is a unitary Vandermonde matrix.

Multiplying by the DFT matrix involves $O(N^2)$ operations. A commonly used algorithm to compute the DFT is the Fast Fourier Transform approach or FFT, which scales as $O(N \log N)$. The FFT algorithm is described in appendix A.2

Once the DFT is performed, we are able to identify the level of energy at each frequency. Let K be the number of frequencies we keep.

The k-th element of the power spectrum associated with a frame f_p is given by :

$$\forall k \in \{1, \ldots, K\} \quad P_p^k = \frac{|\hat{f}_p^k|^2}{N} \tag{3.3.2}$$

4. **Step 4 : Introducing the triangular filters to reduce the dimensionality**

   In order to get the spectral vectors, the final step consists in applying D triangular filters.

   To that end, we first split the frequencies into D equally sized intervals $(I_d)_{1 \leq d \leq D}$. Let $c_1, \ldots, c_D$ be the centers of these intervals. The filters $F_d(.),\ d \in \{1, \ldots, D\}$ have a linear response from 1 at their center frequencies c_d to zero at their adjacent center frequencies c_{d-1} and c_{d+1}, as described by the equation 3.3.3

$$\forall d \in \{1, \dots, D\} \; \forall x \in \mathbb{R} \quad F_d(x) = \begin{cases} 0 & x < c_{d-1} \\ \frac{x - c_{d-1}}{c_d - c_{d-1}} & c_{d-1} \leq x \leq c_d \\ \frac{c_{d+1} - x}{c_{d+1} - c_d} & c_d \leq x \leq c_{d+1} \\ 0 & k > c_{d+1} \end{cases} \tag{3.3.3}$$

5. **Step 5 : Getting the spectral vectors**

   In order to get the final spectral vectors, let us consider the matrix notations associated with the different steps :

   — We consider the matrix $F \in \mathbb{R}^{L \times N}$ where the p-th row corresponds to the frames $f_p = [f_p^1, \dots, f_p^N]$ of length N, for all $p \in \{1, \dots, L\}$. So,

   $$F = \begin{pmatrix} f_1^1 & \dots & f_1^N \\ \vdots & \vdots & \vdots \\ f_L^1 & \dots & f_L^N \end{pmatrix}$$

   — By multiplying each row of F with the window function, we get the matrix $S \in \mathbb{R}^{L \times N}$:

   $$S = \begin{pmatrix} s_1^1 & \dots & s_1^N \\ \vdots & \vdots & \vdots \\ s_L^1 & \dots & s_L^N \end{pmatrix}$$

   — After applying the DFT, we get the matrix $\hat{F} \in \mathbb{R}^{L \times K}$:

   $$\hat{F} = \begin{pmatrix} \hat{f}_1^1 & \dots & \hat{f}_1^K \\ \vdots & \vdots & \vdots \\ \hat{f}_L^1 & \dots & \hat{f}_L^K \end{pmatrix}$$

   — By taking the power spectrum of each Foutier coefficient, we get the power spectrum matrix $P \in \mathbb{R}^{L \times K}$:

   $$P = \begin{pmatrix} P_1^1 & \dots & P_1^K \\ \vdots & \vdots & \vdots \\ P_L^1 & \dots & P_L^K \end{pmatrix}$$

   — Let $H \in \mathbb{R}^{K \times D}$ be the matrix containing the D triangular filters. Applying the triangular filters to the power spectrum matrix is the result of the dot product between the matrix H and the matrix P as follows.

$$\hat{X} = \begin{pmatrix} — & \hat{X}_1 & — \\ \vdots & \vdots & \vdots \\ — & \hat{X}_L & — \end{pmatrix} = PH$$

We get a matrix $\hat{X} \in \mathbb{R}^{L \times D}$. Let $\hat{X}_1, \ldots, \hat{X}_L \in \mathbb{R}^D$ be the rows of the matrix $\hat{X}$.

— After normalizing the vectors $\hat{X}_1, \ldots, \hat{X}_L$, we get the final spectral vectors $X_1, \ldots, X_L \in \mathbb{R}^D$, represented in figure 3.3. In the x-axis, we have the frame index. In the y-axis, we represent the dimensions of the spectral vectors, which correspond the the different frequency bands captured by the triangular filters.

We have $L = 100$ and $D = 26$ in figure 3.3.

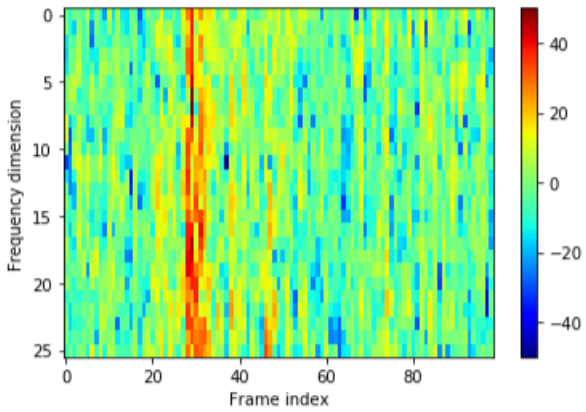


FIGURE 3.3 – The final spectral vectors

3.3.2 Getting the latent spectral vectors using unsupervised representation learning

After extracting the spectral vectors from the daily returns of an asset, we use a conditional variational autoencoder to learn a latent low dimensional space. Conditional Variational Autoencoders (CVAEs) can be regarded as enhanced autoencoders where a Bayesian approach is used to learn the probability distribution of the input data.

Theoretical background

In this section, we give a formal theoretical description of the conditional variational autoencoder.

Introduction

Let $X_1, \ldots, X_N$ be the spectral vectors resulting from the time frequency analysis described in the previous section.

We make the assumption that these vectors are independent and identically distributed according to some complex distribution $p_\theta(x)$.

The objective of the VAE is to model this distribution from a decodable continuous latent space. In terms of architecture, VAEs look similar to the vanilla autoencoders. They are made up of an encoder and a decoder, as shown in figure 3.4.

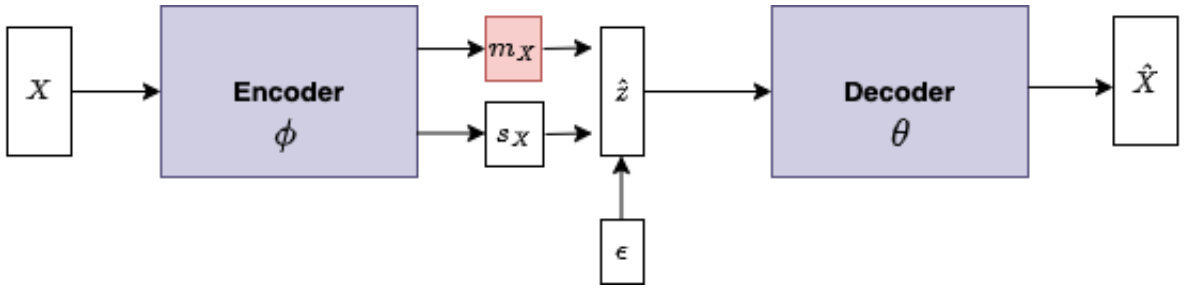


FIGURE 3.4 – Variational Autoencoders architecture

Both models are used to reconstruct the input data X while learning a latent representation. However, unlike vanilla autoencoders, VAEs have a continuous latent space.

In order to incorporate additional information into the latent low dimensional representation of the spectral vectors, we used a conditional variational autoencoder (CVAE) (Sohn et al., 2015). It consists in modifying the VAE architecture by imposing a condition Y on both the encoder and the decoder, as shown in figure 3.5.

In our implementation, the additional information Y is the sign of the return for the corresponding window represented by the spectral vector X.

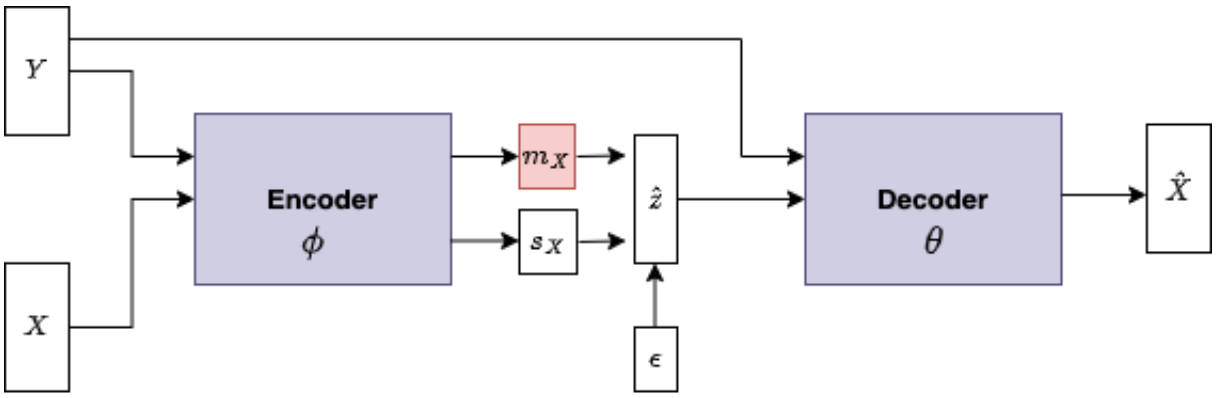


FIGURE 3.5 – A representation of the Conditional Variational Autoencoder structure used.

The CVAE can be seen as a probabilistic model of data based on a continuous mixture of distributions. In other words, we are interested in learning the distribution $p(x|y)$. And for that, we introduce a continuous

latent variable z as follows :

$$p_\theta(x|y) = \int p(x \mid z, y, \theta)p(z)dz$$

Where $p(z) = \mathcal{N}(0, I_d)$ is a continuous prior (d is the dimensionality of the hidden space and D is the dimensionality of the spectral vectors). Obviously, $d << D$.

Setting up the objective function

To train this model, we want to maximize the marginal log-likelihood $\max_\theta \log p(x \mid y, \theta)$ of the dataset. If we marginalize over the latent variables z and consider their distibutions $q(z)$, we obtain a variational lower bound which depends on both the distributions $q(z)$ and θ as follows :

$$\begin{aligned}
\log(p_\theta(x|y)) &= \sum_{i=1}^{N} \log p_\theta(x_i|y_i) \quad (\ X_1, \ldots, X_n \text{ are i.i.d}) \\
&= \sum_{i=1}^{N} \log \left(\int_{z_i} p_\theta(x_i, z_i|y_i) dz_i \right) \\
&= \sum_{i=1}^{N} \log \left(\int_{z_i} \frac{p_\theta(x_i, z_i|y_i) q(z_i)}{q(z_i)} dz_i \right) \\
&\geq \sum_{i=1}^{N} \int_{z_i} q(z_i) \log \left(\frac{p_\theta(x_i, z_i|y_i)}{q(z_i)} \right) dz_i \quad \text{(Jensen Inequality)} \\
&= \sum_{i=1}^{N} \underbrace{\mathbb{E}_{q(z_i)} \left[\log \left(\frac{p_\theta(x_i, z_i|y_i)}{q(z_i)} \right) \right]}_{\mathcal{L}(q(z_i), \theta)}
\end{aligned}$$

The gap ϵ between the marginal log-likelihood and the **variational lower bound** $\sum_{i=1}^{N} \mathcal{L}(q(z_i), \theta)$ is reduced to 0 when $q(z_i) = p_\theta(z_i \mid x_i, y_i)\ \forall i \in \{1, \ldots, N\}$.

$p_\theta(z_i \mid x_i, y_i)$ is called the **posterior distribution**.

Indeed,

$$
\begin{aligned}
\epsilon &= \log(p_\theta(x|y)) - \sum_{i=1}^{N} \mathcal{L}(q(z_i), \theta) \\
&= \sum_{i=1}^{N} \left(\log(p_\theta(x_i|y_i)) - \mathcal{L}(q(z_i), \theta)\right) \\
&= \sum_{i=1}^{N} \left(\int_{z_i} q(z_i) \log(p_\theta(x_i|y_i)) dz_i - \int_{z_i} q(z_i) \log\left(\frac{p_\theta(x_i, z_i|y_i)}{q(z_i)}\right) dz_i\right) \\
&= \sum_{i=1}^{N} \int_{z_i} q(z_i) \left(\log(p_\theta(x_i|y_i)) - \log\left(\frac{p_\theta(x_i, z_i|y_i)}{q(z_i)}\right)\right) dz_i \\
&= \sum_{i=1}^{N} \int_{z_i} q(z_i) \log\left(\frac{p_\theta(x_i|y_i) q(z_i)}{p_\theta(x_i, z_i|y_i)}\right) dz_i \\
&= \sum_{i=1}^{N} \int_{z_i} q(z_i) \log\left(\frac{p_\theta(x_i|y_i) q(z_i)}{p_\theta(z_i|x_i, y_i) p_\theta(x_i|y_i)}\right) dz_i \\
&= \sum_{i=1}^{N} \int_{z_i} q(z_i) \log\left(\frac{q(z_i)}{p_\theta(z_i|x_i, y_i)}\right) dz_i \\
&= \sum_{i=1}^{N} \mathcal{KL}\left(q(z_i) \parallel p_\theta(z_i|x_i, y_i)\right)
\end{aligned}
$$

Therefore, if we want to maximize $\log(p_\theta(x|y))$, we could alternate maximizing the variational lower bound (also called evidence lower bound ELBO) with respect to θ and with respect to q by making $q(z_i) = p_\theta(z_i|x_i, y_i)$ (the posterior distribution). It's the principle of the **Expectation Maximization** algorithm. The E-step consists of maximizing the lower bound with respect to q by taking the expectation with respect to the posterior distribution. The M-step consists in maximizing w.r.t θ.

The posterior distribution $p_\theta(z|x, y)$ is easily calculated in the case of the Gaussian Mixture Model (or the Hidden Markov Model). However, it's intractable in our case since the evidence is hard to compute. It's no longer just a sum over a discrete set of hidden variables.

Variational Inference is a way of solving the problem of approximating an unknown distribution. For that, we choose a family of known distributions $\mathcal{Q}$ and we minimize the $\mathcal{KL}$ divergence between the unknown distribution p^* and the space of parameterized distributions. The larger the family, the smaller the optimal "distance" will be, as shown in figure 3.6.

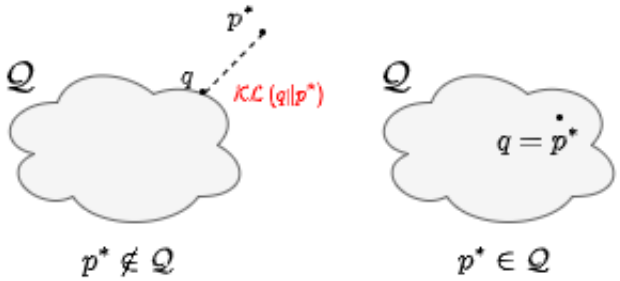


FIGURE 3.6 – Variational Inference illustration

To have a very flexible family $\mathcal{Q}$ of distributions, we use a neural network called the **encoder**.

The learning process consists then in maximizing the variational lower bound while trying to approximate the posterior $p_\theta(z|x,y)$ among a family of distributions $q(z|x,y)$ using the encoder.

So, $\forall i \in \{1,\dots,N\} \quad q(z_i|x_i,y_i) = \mathcal{N}(z_i|m(x_i,y_i,\phi), \text{diag}(s^2(x_i,y_i,\phi))$, ϕ represents all the parameters of the encoder.

To sum up, we will try to maximize the variational lower bound $\sum_{i=1}^{N} \underbrace{\mathbb{E}_{q(z_i)}\left[\log\left(\frac{p_\theta(x_i,z_i|y_i)}{q(z_i)}\right)\right]}_{\mathrm{V}_i}$ with

$q(z_i) = q(z_i|x_i,y_i) = \mathcal{N}(z_i \mid m(x_i,y_i,\phi), \text{diag}(s^2(x_i,y_i,\phi))$

The variational lower bound can be expressed differently as follows :

$$
\begin{aligned}
\forall i \in \{1,\dots,N\} \quad \mathrm{V}_i &= \mathbb{E}_{q(z_i|x_i,y_i)}\left[\log\left(\frac{p_\theta(x_i,z_i|y_i)}{q(z_i|x_i,y_i)}\right)\right] \\
&= \mathbb{E}_{q(z_i|x_i,y_i)}\left[\log\left(\frac{p_\theta(x_i|z_i,y_i)p(z_i)}{q(z_i|x_i,y_i)}\right)\right] \\
&= \mathbb{E}_{q(z_i|x_i,y_i)}\left[\log(p_\theta(x_i|z_i,y_i))\right] + \mathbb{E}_{q(z_i|x_i,y_i)}\left[\log\left(\frac{p(z_i)}{q(z_i|x_i,y_i)}\right)\right] \\
&= \mathbb{E}_{q(z_i|x_i,y_i)}\left[\log(p_\theta(x_i|z_i,y_i))\right] - \mathbb{E}_{q(z_i|x_i,y_i)}\left[\log\left(\frac{q(z_i|x_i,y_i)}{p(z_i)}\right)\right] \\
&= \mathbb{E}_{q(z_i|x_i,y_i)}\left[\log(p_\theta(x_i|z_i,y_i))\right] - \mathcal{KL}\left(q(z_i|x_i,y_i) \parallel p(z_i)\right)
\end{aligned}
$$

Finally, our objective is :

$$\begin{aligned} \max_{\theta,\phi} \quad & \sum_{i=1}^{N} \left(\mathbb{E}_{q(z_i|x_i,y_i)} \left[\log(p_\theta(x_i|z_i,y_i)) \right] - \mathcal{KL}\left(q(z_i|x_i,y_i) \parallel p(z_i) \right) \right) \\ \text{s.t.} \quad & q(z_i|x_i,y_i) = \mathcal{N}(z_i|m(x_i,y_i,\phi), \text{diag}(s^2(x_i,y_i,\phi)) \text{ and} \\ & p(x|z,y,\theta) \text{ depends on the most suitable choice for the data} \end{aligned}$$

As we always prefer to minimize the objective, we rewrite it as follows :

$$\begin{aligned} \min_{\theta,\phi} \quad & \underbrace{\sum_{i=1}^{N} \left(-\mathbb{E}_{q(z_i|x_i,y_i)} \left[\log(p_\theta(x_i|z_i,y_i)) \right] + \mathcal{KL}\left(q(z_i|x_i,y_i) \parallel p(z_i) \right) \right)}_{\textbf{Objective Function}} \\ \text{s.t.} \quad & q(z_i|x_i,y_i) = \mathcal{N}(z_i|m(x_i,y_i,\phi), \text{diag}(s^2(x_i,y_i,\phi)) \text{ and} \\ & p(x|z,y,\theta) \text{ to be specified} \end{aligned}$$

So overall, the model looks as follows : The encoder takes the input x_i and the condition y_i, produces a distribution over latent codes $q(z_i|x_i,y_i)$ which should approximate the posterior distribution $p_\theta(z_i|x_i,y_i)$ (at least after training), samples a point from this distribution $\widehat{z_i} \sim q(z_i|x_i,y_i,\phi)$, and finally feeds it into a decoder that outputs a distribution over the data.

Training the CVAE

We want to perform the following optimization :

$$\begin{aligned} \min_{\theta,\phi} \quad & \underbrace{\sum_{i=1}^{N} \left(-\mathbb{E}_{q(z_i|x_i,y_i)} \left[\log(p_\theta(x_i|z_i,y_i)) \right] + \mathcal{KL}\left(q(z_i|x_i,y_i) \parallel p(z_i) \right) \right)}_{\textbf{Objective Function}} \\ \text{s.t.} \quad & q(z_i|x_i,y_i) = \mathcal{N}(z_i|m(x_i,y_i,\phi), \text{diag}(s^2(x_i,y_i,\phi)) \end{aligned}$$

For that, we use the Stochastic Gradient Descent algorithm. The objective function has an expected value inside, which we will approximate by sampling. And to differentiate through this approximation, we will use the **reparameterization trick**. It consists in replacing the sampling from the distribution $\widehat{z} \sim q(z|x,y,\phi)$ (which depends on the parameter ϕ) by sampling from a deterministic transformation of the standard normal distribution (which doesn't depend on the parameters) : $\varepsilon \sim \mathcal{N}(0,I)$; $\widehat{z} =$

$m(x, y, \phi) + \varepsilon s(x, y, \phi)$. We can then differentiate the loss w.r.t the parameters while taking the sample ε as constant.

The computation of the gradients of $\xi(\theta, \phi) := \sum_{i=1}^{N} \left(-\mathbb{E}_{q(z_i|x_i,y_i)} \left[\log(p_\theta(x_i|z_i, y_i))\right]\right)$ w.r.t θ are detailed in appendix A.3

Introducing the High and Low Turbulence regimes by clustering the latent spectral vectors

The step involving the CVAE has been articulated as follows : For each geographic zone, we collect a universe of large cap stocks (The 500 $S\&P$ index in the US, The Stoxx 600 in Europe, the Nikkei 225 in Japan, and the FTSE 100 in the UK), typically present in the broad geography index of interest. In total, per geography, we tend to have between 200 and 600 time series with typically thirty years of history in each.

From this, we are able to derive the spectral vectors using the time frequency approach. These vectors form the dataset used to estimate the parameters of the CVAE.

Once the CVAE is trained, we use the encoder to map the spectral vectors to their low dimensional representation. The resulting vectors are called "latent sepctral vectors".

In order to evaluate the clustering properties of the approach, we use a Gaussian Mixture Model to cluster the latent spectral vectors extracted from all the stocks composing the main indexes from the different geographies into two different clusters.

We run the process in each geography and we end up with two well separated clusters : The High Turbulence cluster and the Low Turbulence cluster. We compare the distributions of the daily returns in each cluster in terms of first moments (μ), standard deviations (σ) and Sharpe ratios (sr). The following table summarizes the results :

The index l refers to the low risk cluster and the index h refers to the high risk cluster. The mean is expressed with a factor ($\times 10^{-4}$), the standard deviation with a factor($\times 10^{-3}$) and the Sharpe ratio with a factor ($\times 10^{-2}$).

Geography	μ_l	μ_h	σ_l	σ_h	sr_l	sr_h
US	6.05	3.55	8.58	17.3	7.05	2.04
Europe	5.57	1.82	8.48	17.1	6.56	1.06
Japan	3.95	2.15	9.38	17.3	4.21	1.23
UK	5.24	1.84	8.01	16.1	6.54	1.14

TABLE 3.1 – Moments of the high and low risk distributions of the daily returns

We can conclude that the latent spectral vectors exhibit good clustering propoerties, which suggests the

definition of two regimes : The High Turbulence regime and the Low Turbulence regime. These regimes are well separated and tend to be very stable across time, which suggests the use of a Hidden Markov Model in order to introduce the sequentiality in the mixture model.

3.3.3 Learning the dynamics of the latent spectral vectors

Introduction

Let us consider a sequence of T spectral vectors $(X_1, \dots, X_T)$ associated with an index in one of the geographies under consideration. We denote $(\tilde{X}_1, \dots, \tilde{X}_T)$ the sequence of latent spectral vectors resulting from the transformation of $(X_1, \dots, X_T)$ through the CVAE encoder.

The objective of this section is to model the dynamics of the sequence $(\tilde{X}_1, \dots, \tilde{X}_T)$ using a Hidden Markov Model (HMM), where the hidden states of the HMM correspond to the high and the low turbulence states.

Section 3.3.3 gives a brief description of the parameterization of the HMM graphical model. Section 3.3.3 deals with the inference problem and introduces the filtering and smoothing probabilities. In section 3.3.3, we present the learning process using the Expectation Maximization algorithm. Finally, in section 3.3.3, we predict the probability of being in a low turbulence state over the next period of time, which is the final output of the Low Turbulence approach.

The parameterization of the graphical model

The hidden state at time t is denoted by H_t and the observation (in our case, the latent spectral vector) at time t by $\tilde{X}_t$. Let us assume there are $M = 2$ possible hidden states (corresponding to the low and the high turbulence regimes) and that the observations are continuous in $\mathbb{R}^d$. Let us also suppose we have T continuous observations $(\tilde{X}_t)_{1 \leq T} \in \mathbb{R}^{T \times d}$

Figure 3.7 shows the difference between the graphical representation associated with the vanilla Gaussian mixture model used in the clustering section 3.3.2 and the graphical model associated with the HMM.

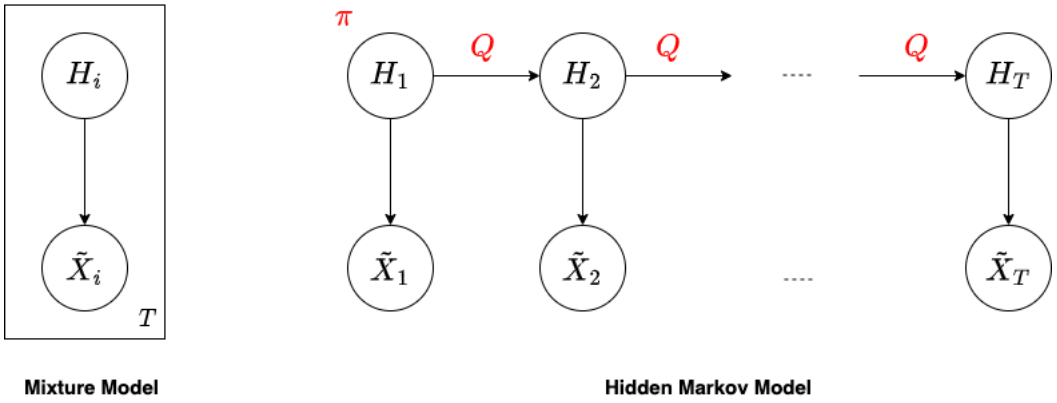


FIGURE 3.7 – Comparing the graphical representations of the mixture model and the HMM

A Gaussian Mixture model would be parameterized by a vector $\pi = (\pi_1, \ldots, \pi_M) \in \mathbb{R}^M$ such that $\sum_{m=1}^{M} \pi_m = 1$ and $(\mu_m, \Sigma_m) \in \mathbb{R}^d \times \mathbb{R}^{d \times d}$ for each $m \in \{1, \ldots, M\}$, such that :

$$H_i \sim \mathcal{M}(1, \pi_1, \ldots, \pi_M)$$
$$\forall t \in \{1, \ldots, T\} \; \tilde{X}_t | H_t = m \sim \mathcal{N}_d(\mu_m, \Sigma_m)$$

In the HMM graphical model, each vertical slice represents a time step. Applying d-separation to the the graphical model, we can retrieve the well known result that the future is independent of the past given the present. [1].

To parameterize the Hidden Markov Model, we need to assign local conditional probabilities to each of the nodes. We represent the state at time t as a multinomial random variable H_t with **M** hidden states.

The first state node has no parents, thus we endow this node with an unconditional distribution π (called the **intital state distribution**), such that :

$$\forall i \in [\![1, M]\!] \quad \pi_i = p(H_0 = i)$$

Each successive state node has the previous state node in the chain as its parent, thus we need a $M \times M$ matrix to specify its local conditional probability. We define a **state transition matrix** Q, where the (i, j)th entry Q_{ij} of Q is defined to be the transition probability $p(H_{t+1} = j | H_t = i)$. We assume a

1. By present, we mean conditioning on the state note H_t, not the output node $\tilde{X}_t$

homogeneous HMM, so the transition probability is independent of t.

Each of the output node has a single state node as a parent, thus we require a probability distribution $p(\tilde{X}_t|h_t)$ called the **emission distribution**. We assume the emission distribution to be Gaussian and independent of t.

An emission at time t is represented by a random vector $\tilde{X}_t \in \mathbb{R}^d$. In the rest of the paper, we will consider Gaussian emissions.

Therefore, the parameterization can be summarized as follows :

$$\begin{aligned} \forall m \in \{1,\dots,M\} \quad & \pi_m = p(H_1 = m) \\ \forall h, h' \in \{1,\dots,M\} \quad & Q_{h,h'} = p(H_{t+1} = h'|H_t = h) \\ \forall t \in \{1,\dots,T\}\ \forall h \in \{1,\dots,M\} \quad & \tilde{X}_t|H_t = h \sim \mathcal{N}_d(\mu_h, \Sigma_h) \end{aligned}$$

We denote $\mu = (\mu_m)_{m\in[\![1,M]\!]} \in \mathbb{R}^{M\times d}$ and $\Sigma = (\Sigma_m)_{m\in[\![1,M]\!]} \in \mathbb{R}^{M\times d\times d}$ the parameters of the emission distributions.

The parameters associated with an HMM are : $\theta = (\pi, Q, \mu, \Sigma)$.

The Inference problem

1. **The joint probability**

   For a particular sequence $(\mathbf{h}, \tilde{\mathbf{x}}) = (h_1,\dots,h_T,\tilde{x}_1,\dots,\tilde{x}_t)$, we obtain the following joint probability :

   $$p_\theta(\mathbf{h}, \tilde{\mathbf{x}}) = p(h_1)\prod_{t=1}^{T-1} p(h_{t+1}|h_t)\prod_{t=1}^{T} p(\tilde{x}_t|h_t) \tag{3.3.4}$$

   To introduce Q and π into this equation and adopt a notation in which state variables can be used as indices, we use the one hot encoding notation : The one hot vector $\tilde{h}_t \in \{0,1\}^M$ is defined as follows :

   $$\forall t \in [\![1,T]\!]\ \forall i \in [\![1,M]\!] \quad \tilde{h}_t^i = \mathbb{1}_{\{i=m\}} \iff h_t = m \tag{3.3.5}$$

   We can then define $Q_{h_t,h_{t+1}}$ and π_{h_0} as follows :

   $$Q_{h_t,h_{t+1}} := \prod_{i,j=1}^{M} [Q_{ij}]^{\tilde{h}_t^i \tilde{h}_{t+1}^j} \quad \text{and} \quad \pi_{h_0} := \prod_{i=1}^{M} [\pi_i]^{\tilde{h}_0^i} \tag{3.3.6}$$

Similarly, we define μ_{h_t} and Σ_{h_t} as follows :

$$\mu_{h_t} := \left(\prod_{i=1}^{M} [\text{diag}(\mu_i)]^{\tilde{h}_t^i} \right) \mathbb{1}_D \quad \text{and} \quad \Sigma_{h_t} := \prod_{i=1}^{M} [\Sigma_i]^{\tilde{h}_t^i}$$

Plugging the definitions into the joint probability, we obtain the parameterized distribution :

$$p_\theta(\mathbf{h}, \tilde{\mathbf{x}}) = \pi_{h_1} \prod_{t=1}^{T-1} Q_{h_t, h_{t+1}} \prod_{t=1}^{T} \mathcal{N}(\tilde{x}_t; \mu_{h_t}, \Sigma_{h_t}) \tag{3.3.7}$$

Hence,

$$p_\theta(\tilde{\mathbf{x}}) = \sum_{h_1} \sum_{h_2} \cdots \sum_{h_T} \pi_{h_1} \prod_{t=1}^{T-1} Q_{h_t, h_{t+1}} \prod_{t=1}^{T} \mathcal{N}(\tilde{x}_t; \mu_{h_t}, \Sigma_{h_t}) \tag{3.3.8}$$

2. **Smoothing - Filtering probabilities**

We introduce the following probabilities :

— **Filtering probabilities :** $\xi \in \mathbb{R}^{T \times M}$

$$\xi(t,h) := p(H_t = h | \tilde{X}_1 = \tilde{X}_1, \ldots, \tilde{X}_t = \tilde{X}_t) \quad \forall (t,h) \in [\![1,T]\!] \times [\![1,M]\!]$$

— **Smoothing probabilities :** $\psi \in \mathbb{R}^{T \times M}$ and $\phi \in \mathbb{R}^{T-1 \times M \times M}$

$$\psi(t,h) := p(H_t = h | \tilde{X}1 = \tilde{X}1, \ldots, \tilde{X}_t = \tilde{X}_t) \quad \forall (t,h) \in [\![1,T]\!] \times [\![1,M]\!]$$
$$\phi(t,h,h') := p(H_t = h, H_{t+1} = h' | \tilde{X}_1 = \tilde{X}_1, \ldots, \tilde{X}_t = \tilde{X}_t)$$
$$\forall (t,h,h') \in [\![1,T-1]\!] \times [\![1,M]\!] \times [\![1,M]\!]$$

In order to compute them and the likelihood efficiently, we will use the Forward Backward algorithm 3, as explained in appendix A.4.

Algorithm 3 Forward Backward Algorithm

Input: Observations $\tilde{X}_1 \dots \tilde{X}_T$
Output: $(\xi_t)_{1\leq t\leq T}$ and $(\psi_t)_{1\leq t\leq T}$ (The filtering and smoothing probabilties)

$c_1 \leftarrow \mathbb{1}^T \Gamma(1)\pi$
$\xi_1 \leftarrow \Gamma(1)\pi / c_1$
for $t \leftarrow 2, \dots, T$ **do**
    $\tilde{\xi}_t \leftarrow \Gamma(t) Q^T \xi_{t-1}$
    $c_t \leftarrow \mathbb{1}^T \tilde{\xi}_t$
    $\xi_t \leftarrow \tilde{\xi}_t / c_t$
end for
$\tilde{\beta}_T \leftarrow 1/c_T$
for $t \leftarrow 1, \dots, T-1$ **do**
    $\tilde{\beta}_{T-t} \leftarrow Q\Gamma(T-t+1)\tilde{\beta}_{T-t+1}$
    $\psi_{T-t} \leftarrow \text{diag}(\xi_{T-t}) Q \Gamma(T-t+1) \text{diag}(\tilde{\beta}_{T-t+1})$
    $\phi_{T-t} \leftarrow \psi_{T-t} \mathbb{1}_M$
end for=0

Learning the parameters of the HMM using the Expectation Maximization (EM) Algorithm

In order to learn the parameters of the HMM, we use the Expectation Maximization algorithm.

1. **Introducing the EM algorithm**

   The EM algorithm is an iterative method for finding maximum likelihood estimates of parameters in statistical models, where the models depend on unobserved latent variables.

   Consider for instance N observations $a_1, \dots, a_N$ and the latent variables associateed with them $z_1, \dots, z_N$.

   We assume the pairs (a_i, z_i) to be independent and identically distributed.

   For $(a, z) = (a_1, z_1, \dots, a_N, z_N)$, the objective is to maximize :

$$\log(p(a;\theta)) = \sum_{i=1}^{N} \log\left(\sum_{z_i} p(a_i, z_i; \theta)\right)$$

   By conditioning on a latent variable z and using the Jensen inequality, we get :

$$\begin{aligned}\log(p(a;\theta)) &= \log\left(\sum_z p(a,z;\theta)\right)\\ &= \log\left(\sum_z q(z)\frac{p(a,z;\theta)}{q(z)}\right)\\ &\geq \sum_z q(z)\log\left(\frac{p(a,z;\theta)}{q(z)}\right)\\ &= \underbrace{\mathbb{E}_q[\log(p(a,z;\theta))] + f(q)}_{\mathcal{L}(q,\theta)}\end{aligned}$$

With equality iff $q(z) = p_\theta(z|a)$

The EM algorithm can then be summarized as depicted in 4

Algorithm 4 EM algorithm

Input: Observations $a_1 \dots a_N$
Output: Optimal θ

1: Initialize θ
2: $\xi_1 \leftarrow \Gamma(1)\pi/c_1$
3: **while** (Not converged) **do**
4: E-step : $q(z) = p(z|a;\theta^{(i-1)})$
5: M-step : $\theta^{(i)} = \arg\max_\theta \mathbb{E}_q[\log(p(a,z;\theta)]$
6: **end while**

2. **Learning the parameters of the HMM using the EM algorithm**

(a) **The E-step**

At the iteration i, the complete loglikelihood $\log(p_{\theta_i}(\mathbf{h},\tilde{\mathbf{x}}))$ is expressed as follows :

$$\log(p_{\theta_i}(\mathbf{h},\tilde{\mathbf{x}})) = \log\left(\pi_{h_1}^{(i)}\prod_{t=1}^{T-1} Q_{h_t,h_{t+1}}^{(i)}\prod_{t=1}^{T}\mathcal{N}(\tilde{x}_t;\mu_{h_t}^{(i)},\Sigma_{h_t}^{(i)})\right) \tag{3.3.9}$$

$$= \log(\pi_{h_1}^{(i)}) + \sum_{t=1}^{T-1}\log(Q_{h_t,h_{t+1}}^{(i)}) + \sum_{t=1}^{T-1}\log(\mathcal{N}(\tilde{x}_t;\mu_{h_t}^{(i)},\Sigma_{h_t}^{(i)})) \tag{3.3.10}$$

The E-step consists in computing : $\mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(p_{\theta_i}(\mathbf{h},\tilde{\mathbf{x}}))]$.

We have :

$$\mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(\pi^{(i)}_{h_1})] = \sum_{h=1}^{M} \log(\pi^{(i)}_h) p(H_1 = h|\tilde{\mathbf{x}}) \tag{3.3.11}$$

$$\mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(Q^{(i)}_{h_t,h_{t+1}})] = \sum_{h=1}^{M}\sum_{h'=1}^{M} \log(Q^{(i)}_{hh'}) p(H_t = h, H_{t+1} = h'|\tilde{\mathbf{x}}) \tag{3.3.12}$$

$$\mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(\mathcal{N}(\tilde{X}_t; \mu^{(i)}_{h_t}, \Sigma^{(i)}_{h_t}))] = \sum_{h=1}^{M} \log(\mathcal{N}(\tilde{X}_t; \mu^{(i)}_h, \Sigma^{(i)}_h)) p(h_t = h|\tilde{\mathbf{x}}) \tag{3.3.13}$$

By summing the three parts, we obtain the following expression :

$$\begin{aligned}\mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(p_{\theta_i}(\mathbf{h}, \tilde{\mathbf{x}}))] = &\sum_{h=1}^{M} \log(\pi^{(i)}_h) p(H_1 = h|\tilde{\mathbf{x}}) + \sum_{t=1}^{T-1}\sum_{h=1}^{M}\sum_{h'=1}^{M} \log(Q^{(i)}_{hh'}) p(H_t = h, H_{t+1} = h'|\tilde{\mathbf{x}}) \\ &+ \sum_{t=1}^{T-1}\sum_{h=1}^{M} \log(\mathcal{N}(\tilde{X}_t; \mu^{(i)}_h, \Sigma^{(i)}_h)) p(H_t = h|\tilde{\mathbf{x}})\end{aligned} \tag{3.3.14}$$

Introducing the smoothing probabilities, we obtain :

$$\boxed{\begin{aligned}\mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(p_{\theta_i}(\mathbf{h}, \tilde{\mathbf{x}}))] = &\sum_{h=1}^{M} \log(\pi^{(i)}_h) \psi(1, h) + \sum_{t=1}^{T-1}\sum_{h=1}^{M}\sum_{h'=1}^{M} \log(Q^{(i)}_{hh'}) \phi(t, h, h') \\ &+ \sum_{t=1}^{T-1}\sum_{h=1}^{M} \log(\mathcal{N}(\tilde{X}_t; \mu^{(i)}_h, \Sigma^{(i)}_h)) \psi(t, h)\end{aligned}} \tag{3.3.15}$$

(b) **The M-step**

The objective is to maximize $\mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(p_\theta(\mathbf{h}, \tilde{\mathbf{x}}))]$ with respect to θ :

$$\theta_{i+1} = \arg\max_{\theta} \mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(p_\theta(\mathbf{h}, \tilde{\mathbf{x}}))] \tag{3.3.16}$$

As shown in the appendix A.5, we obtain the following update equations, expressed with the smoothing probabilities.

$\forall (h, h') \in [\![1, M]\!] \times [\![1, M]\!]$

$$\pi_h^{(i+1)} = \psi(1,h) \tag{3.3.17}$$

$$Q_{h,h'}^{(i+1)} = \frac{\sum_{t=1}^{T} \phi(t,h,h')}{\sum_{t=1}^{T} \psi(t,h)} \tag{3.3.18}$$

$$\mu_h^{(i+1)} = \frac{\sum_{t=1}^{T} \psi(t,h)\tilde{X}_t}{\sum_{t=1}^{T} \psi(t,h)} \tag{3.3.19}$$

$$\Sigma_h^{(i+1)} = \frac{\sum_{t=1}^{T} \psi(t,h)(\tilde{X}_t - \mu_h^{(i)})(\tilde{X}_t - \mu_h^{(i)})^T}{\sum_{t=1}^{T} \psi(t,h)} \tag{3.3.20}$$

Predicting the turbulence state over the next period of time

Once the model is trained using the EM algorithm, we would like to compute the probability of being in a low turbulence state over the next period of time, as shown in figure 3.8.

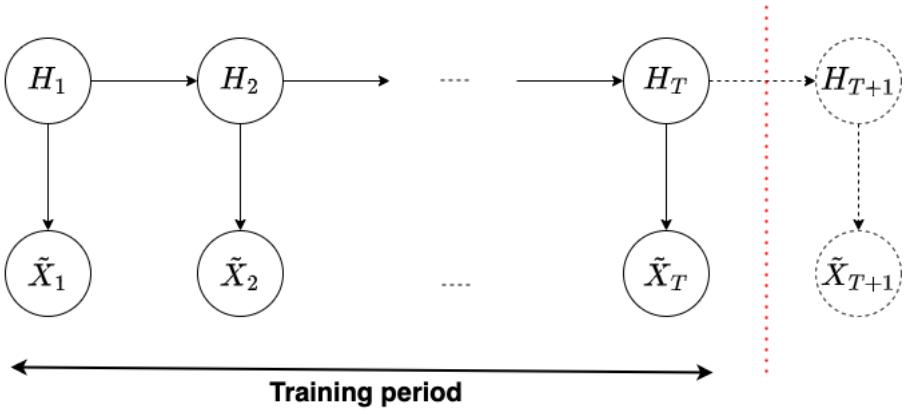


FIGURE 3.8 – Predicting the turbulence state over the next period of time

The prediction over the next period can be calculated as follows :

$$\begin{aligned}\forall h \in [\![1, M]\!] \quad & p(H_{T+1} = h | \tilde{X}_1 = \tilde{x}_1, \dots, \tilde{X}_T = \tilde{x}_T) \\ & = \sum_{h'=1}^{M} p(H_{T+1} = h, H_T = h' | \tilde{X}_1 = \tilde{x}_1, \dots, \tilde{X}_T = \tilde{x}_T) \\ & = \sum_{h'=1}^{M} \underbrace{p(H_{T+1} = h | H_T = h')}_{=Q_{h'h}} \underbrace{p(H_T = h' | \tilde{X}_1 = \tilde{x}_1, \dots, \tilde{X}_T = \tilde{x}_T)}_{\xi(T,h')}\end{aligned}$$

The filtering probabilities $\xi(t, h')$ are calculated using the Forward Backward algorithm as explained before.

3.4 Empirical results

3.4.1 Summarizing the Low Turbulence approach

The Low Turbulence approach is trained on different granularity levels in order to blend a geographical index and bonds.

As the conditional variational autoencoder needs to be trained on a large amount of data, we use the single stocks composing the index as the training data. The cvae model is retrained on an annual basis.

Once the cvae is trained, we use the encoder to map the spectral vectors of the MSCI sectors sub-indices to their low-dimensional representation called the **latent spectral vectors**.

For each sector sub-index, a graphical model is used to learn the dynamics of the latent spectral vectors and infer the probability of being in a low turbulence regime over the next period of time conditioned on the past latent spectral vectors.

The final allocation is based on the aggregation of all these probabilities as shown in figure 3.9

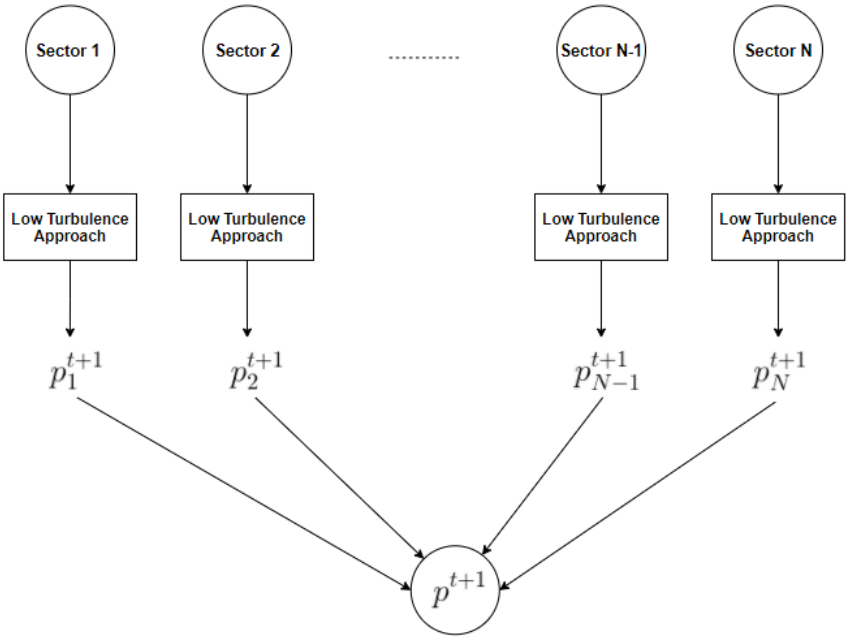


FIGURE 3.9 – Aggregating the predictions of being in the Low Turbulence State over the next period for each particular sector in order to get the final prediction

3.4.2 Risk Control to build the Low Turbulence Indices

The risk control framework aims at reducing the volatility by allocating between the underlying raw strategy and cash. In periods of high volatility, we reduce the weight allocated to the underlying strategy, while in periods of low volatility, all the weight is allocated to the index.

The mechanism for computing the level of volatility of the raw strategy is called EWMA (Exponentially Weighted Moving Average), i.e, a moving average with a rapid memory decay factor.

At the end of this structuring step, we obtain a family of Low Turbulence Indices.

3.4.3 Low Turbulence Downside Protection

Figure 3.10 and 3.11 present the downside protection offered by the Low Turbulence approach for different underlying reference indices during the period 2003-2020.

In order to look at a fair comparison , we take the reference equity index which we submit to the same volatility control of 7% with a decay factor of 0.5.

Both the volatility and the maximum drawdown are drastically reduced using the Low Turbulence approach, across all geographies, in comparison with the underlying index and the 70/30 benchmark with or without volatility control.

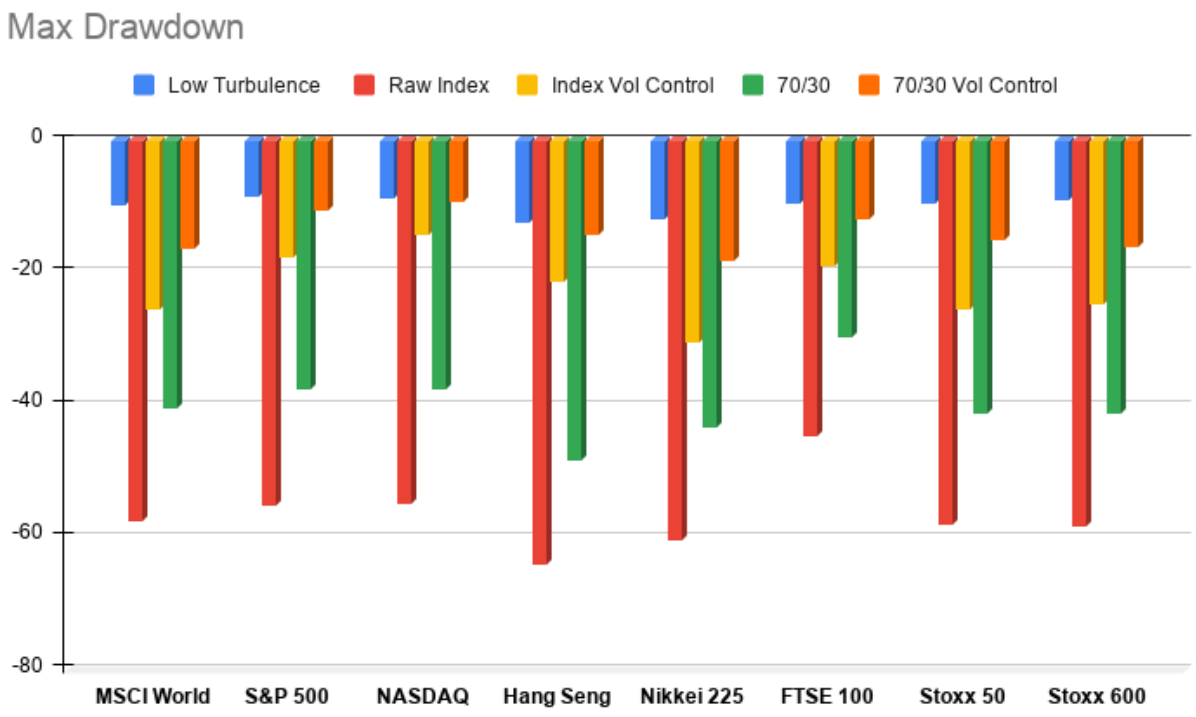


FIGURE 3.10 – The Low Turbulence indices Max Drawdowns, compared to the underlying indices and the 70/30 benchmark with and without volatility control.

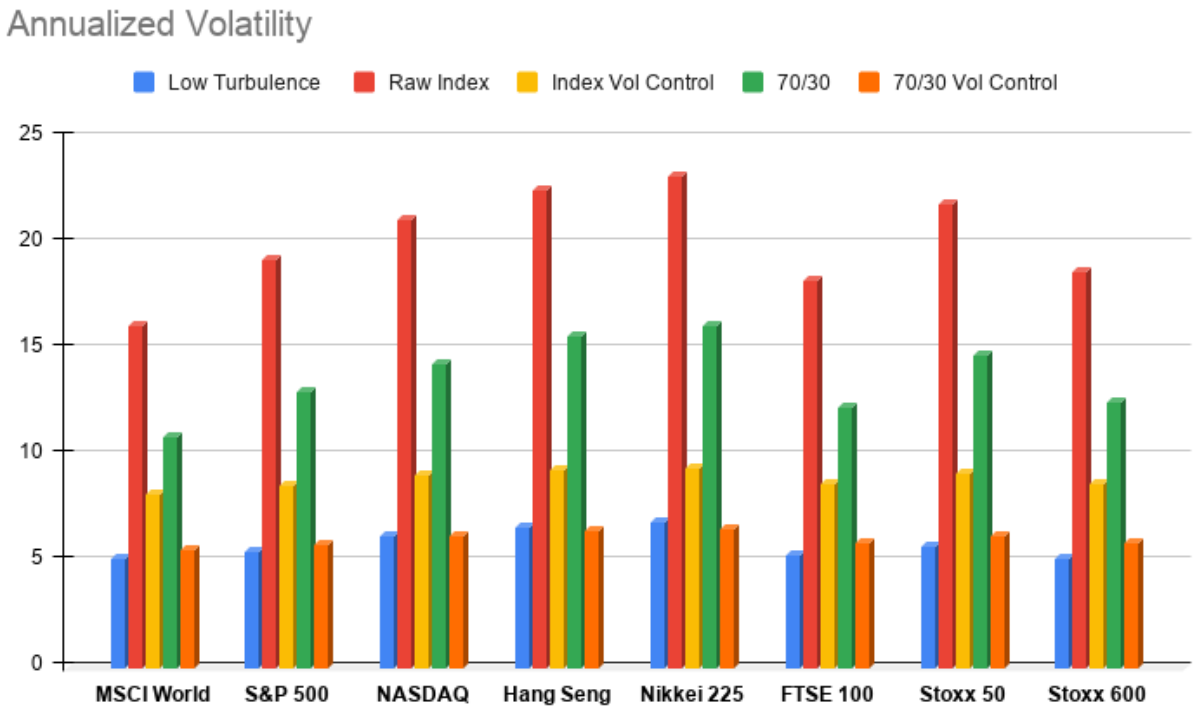


FIGURE 3.11 – The Low Turbulence indices Annualized Volatility, compared to the underlying indices and the 70/30 benchmark with and without volatility control.

3.4.4 Low Turbulence Return/Risk characteristics

Table 3.2 presents the Sharpe ratios of the Low Turbulence approach, compared to the underlying index and the 70/30 benchmark, with or without the volatility control during the period 2003-2020.

	MSCI World	S&P 500	NASDAQ	Hang Seng	Nikkei 225	FTSE 100	Stoxx 50	Stoxx 600
Low Turbulence	**1.37**	**1.35**	**1.2**	**0.89**	**0.85**	**1.1**	**1.01**	**1.26**
Raw Index	0.59	0.54	0.63	0.43	0.37	0.37	0.26	0.38
Index Vol Control	0.92	0.79	0.77	0.55	0.5	0.37	0.33	0.52
70/30	0.78	0.71	0.79	0.57	0.5	0.55	0.43	0.57
70/30 Vol Control	1.15	1	0.94	0.68	0.63	0.57	0.5	0.72

TABLE 3.2 – The Low Turbulence indices Sharpe ratios, compared to the underlying indices and the 70/30 benchmark with and without volatility control.

3.5 Conclusion

The Low Turbulence indices rebalance dynamically between equity and fixed income according to the Low Turbulence forecast.

By reducing the investable periods for any asset to the low turbulence periods only and alternatively invest in safe assets (typically cash or treasuries) during the high turbulence periods, we manage to generate reasonable returns by considerably reducing the volatility and the drawdowns.

As a result, the approach enables the investors to enjoy smooth investment journeys by offering a strong downside protection in addition to significantly higher Sharpe ratios across all geographies.

4 | Bayesian Approach with Attention Mechanism for Learning to Rank Financial Assets

Learning-to-rank algorithms, combined with feed-forward neural networks, have been recently introduced in portfolio construction. Although these neural networks have been successfully used to capture nontrivial relationships in complex systems, they are not well adapted to sequential data and fail to capture the alignment between the input and the output sequences. They can only perform a perception task, which involves learning a mapping between the inputs and the outputs. In order to overcome this limitation, we propose a two-fold ranking system that uses selective memory based on attention mechanisms in order to perform sequential reasoning. Our first processing step is a Bayesian approach mainly used in contest theory. It enables us to construct skill vectors for different assets based on raw prices. An Attention based Recurrent Neural Network is then applied to selectively focus on relevant parts of the skill vectors in order to predict the ranking of a group of assets over the next period of time. On the basis of the aforementioned ranking system, we propose a robust stock picking strategy for large cap US stocks that outperforms both the market and classic momentum strategies.

4.1 Introduction

The practice of quantitative trading, which involves applying scientific methods to the construction of asset portfolios, has now become well established in the financial industry. Momentum strategies fall under this category since they use the positive auto-correlation structure of price returns in order to forecast the future price of an asset. Machine learning algorithms have been used to improve portfolio construction over the last ten years. Several models have been used to predict the direction or the value of the stock market return such as Hidden Markov Models (Y. Zhang, 2004) (Gupta & Dhingra, 2012), Support Vector Machines (L. Cao & Tay, 2001) (K.-j. Kim, 2003), Feed-forward Neural Networks (G. Zhang, Patuwo, & Hu, 1998) (Huang, Lai, Nakamori, Wang, & Yu, 2007), Recurrent Neural Networks (Nelson, Pereira, & De Oliveira, 2017), Ensemble methods (see e.g (Nti, Adekoya, & Weyori, 2020) for a survey), Tree based Algorithms (Basak, Kar, Saha, Khaidem, & Dey, 2019) or more traditional algorithms such as Regression (Freyberger, Neuhierl, & Weber, 2020). A comprehensive literature review on the use of deep learning to predict financial outcomes can be found at (Sezer, Gudelek, & Ozbayoglu, 2020). Recent research has shown the effectiveness of ranking machine learning methods for asset data (X. Zhang, Wu, & Chen, 2022) (Poh, Lim, Zohren, & Roberts, 2021).

The paper proposes a ranking system that combines Bayesian Graphical Inference and Attention-based Recurrent Neural Networks. Two steps are involved in our approach : First, we transform the series of price data into skill vectors using the TrueSkill Algorithm (Herbrich et al., 2006) with Optimal Quantized Belief Propagation (Chichportich, Elie, Kharroubi, & De Servigny, 2020). The resulting skill vectors are then fed into an Attention-Based Recurrent Neural Network in order to predict future asset rankings by

performing sequential reasoning.

The paper is organized as follows : Section 4.2 discusses the literature related to learning to rank in the financial industry. Section 4.3 describes the framework of learning to rank. Section 4.4 describes the ranking system and the optimization process. Finally, section 4.5 compares the results of a strategy based on the proposed ranking system with classic momentum strategies.

4.2 Related Work

Momentum Strategies There are two types of momentum strategies : univariate time series momentum and cross-sectional time series momentum.

Univariate time series momentum involves trading rules based solely on historical price behavior. Time series momentum approaches define signals based on assets' absolute past performance. It was first proposed by (Moskowitz, Ooi, & Pedersen, 2012) over various instruments such as equity, currency, commodity and bond. This pioneering work has led to a rapid growth in literature. As an example, (Georgopoulou & Wang, 2017) examines time series momentum in different asset classes and geographical regions, while (Lim, Wang, & Yao, 2018) examines momentum over the entire past century. High Frequency Data was also studied in (Gao, Han, Li, & Zhou, 2018).

In the Cross Sectional version, assets are compared based on their relative performance. A subset of instruments is selected by sorting them by their momentum value. The approach is based on the work of (Jegadeesh & Titman, 1993), which describes a method of buying assets with better past performance (winners) and selling those with poorer performance (losers). Based on (Jegadeesh & Titman, 1993), this strategy generates significant positive returns. An exhaustive study of a related strategy has been conducted in (Asness, Moskowitz, & Pedersen, 2013) and (Menkhoff, Sarno, Schmeling, & Schrimpf, 2012). An historical assessment of the profitability of cross sectional momentum has been conducted in (Geczy & Samonov, 2016). In (Baz, Granger, Harvey, Le Roux, & Rattray, 2015), a practitioner's approach is described. Several studies have defined similar momentum measures, such as (Blitz, Huij, & Martens, 2011) and (Barroso & Santa-Clara, 2015).

Learning to Rank in Finance The topic of ranking in finance has been discussed recently. It appears that the first research article on this topic is (Song, Liu, & Yang, 2017) in which authors design a stock portfolio selection approach using learning to rank algorithms and sentiment indicators. RankNet approach (Burges et al., 2005) and ListNet one's (Z. Cao, Qin, Liu, Tsai, & Li, 2007) are presented as loss functions trained by a feedforward neural network. Recently, a similar approach has been developed in (L. Wang, 2018) for High Frequency Data. (X. Zhang et al., 2022) proposes a new listwise approach

adapted to a long short stock strategy called ListFold. (Poh et al., 2021) focuses on the same problem with conventional ranking losses. These two articles use feed-forward neural networks to achieve satisfying results directly on price data or factors. Our approach combines a graphical model to process the raw price data with an attention-based recurrent neural network to predict the ranking of assets over the next period. As far as we know, this learning to rank approach has never been used.

4.3 Learning to rank

4.3.1 Brief history of Learning to Rank

The rapid development of the web has made information retrieval a challenge for search engine companies. To provide the best experience to anyone, they build fast and efficient algorithms to rank documents or websites in response to any query. In the past, several approaches have been proposed, including the pointwise approach (Fuhr, 1989), pairwise approach (Joachims, 2002), and listwise approach. Among them, there is a difference in the learning process, specifically the loss function used.

Initially, a pointwise approach was developed. During the training procedure, it considers one document at a time and trains a classifier or regression model to predict how relevant the document is to the query. Classical Machine Learning algorithms are used. For instance, we refer to (Nallapati, 2004) for a SVM-Based Ranking task, to (Gey, 1994) for a Logistic Regression-Based ranking task and to (P. Li, Wu, & Burges, 2007) for a Boosting Tree Based Ranking. Pairwise approaches considers a pair of documents at a time in the training procedure. Therefore, the ranking model predicts relative order. There are many pairwise ranking algorithms proposed in the literature. For instance, we refer to (Burges et al., 2005) or to (Burges, Ragno, & Le, 2006) for Neural Networks Based Ranking, to (Herbrich, Graepel, Obermayer, et al., 2000) for a SVM-Based Ranking task and to (Freund, Iyer, Schapire, & Singer, 2003) for a Boosting Based Ranking. In recent years, the listwise approach has been proposed. For a query, it treats all documents and predicts how they should be ordered.

4.3.2 Introducing the Framework of learning to rank

In this sequel, we give a general description, as described in (Z. Cao et al., 2007), of the listwise learning to rank framework.

We are given a set of N queries $\mathcal{Q} = \{q_1, \dots, q_N\}$. Each query $q_i \in \mathbb{R}^q$ is associated with a set of m documents $d_i = \left(d_i^1, \dots, d_i^m\right) \in \mathbb{R}^{m \times d}$ that we would like to rank according to their relevance to the query.

For each $j \in \{1, \dots, m\}$, the function ψ is used to map the pair (q_i, d_i^j) into the feature vector $x_i^j \in \mathbb{R}^D$.

The sequence of features vectors $x_i = (x_i^1, \ldots, x_i^m) \in \mathbb{R}^{m \times D}$ is then fed into a ranking function f_θ, parameterized by θ, in order to generate m predicted scores $\hat{y}_i = \left(\hat{y}_i^1, \ldots, \hat{y}_i^m\right)$. The score $\hat{y}_i^j$ (for $j \in \{1, \ldots, m\}$) represents the relevance of the document d_i^j to the query q_i.

Based on the order relation in real numbers, it is straightforward to derive a ranking from the predicted scores $\left(\hat{y}_i^1, \ldots, \hat{y}_i^m\right)$.

Moreover, each set of documents $d_i = \left(d_i^1, \ldots, d_i^m\right)$ is associated with a set of true scores $y_i = \left(y_i^1, y_i^2, \ldots, y_i^m\right)$ where $y_i^{(j)}$ (for $j \in \{1, \ldots, m\}$ is the score representing the true relevance of the document $d_i^{(j)}$ to the query q_i.

The training dataset is then composed of the pairs $\mathcal{T} = \{(\hat{y}_i, y_i)\}_{1 \leq i \leq N}$

A representation of the framework is described in figure 4.1.

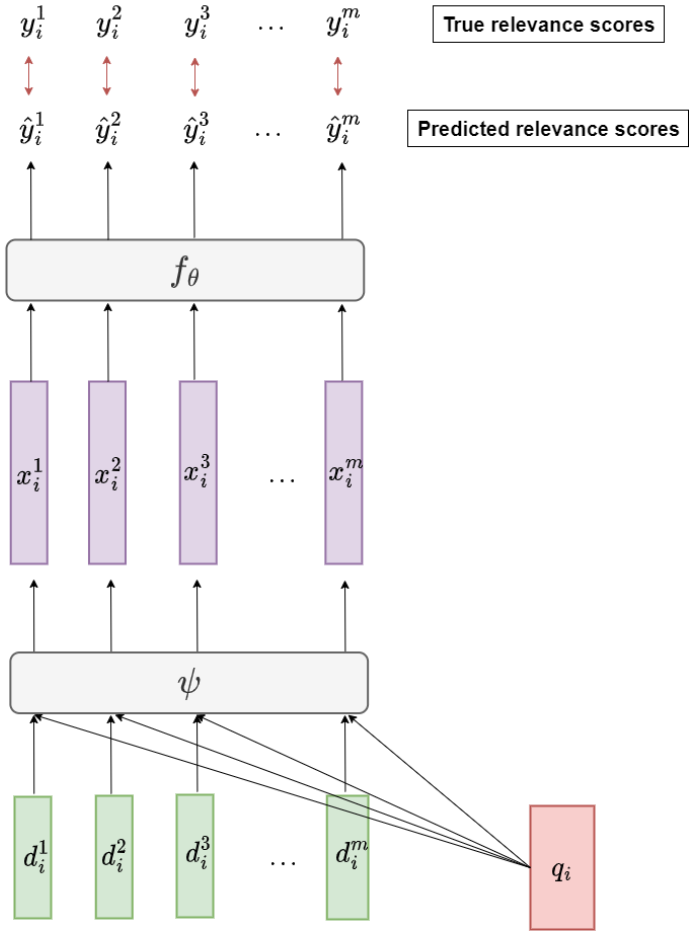


FIGURE 4.1 – Learning to Rank Framework

The objective of learning to rank is to minimise a loss function $\mathcal{L}_\theta$ comparing all the pairs $(\hat{y}_i, y_i)_{1 \leq i \leq N}$ in the training dataset :

$$\min_{\theta\in\theta} \underbrace{\frac{1}{N}\sum_{i=1}^{N} l(\hat{y}_i, y_i)}_{=\mathcal{L}_\theta} \tag{4.3.1}$$

The function $\mathcal{L}_\theta$ can be defined in a variety of ways. The next section introduces the listwise approach, focusing on the ListNet (in 4.3.3) and the RankCosine (in 4.3.3) loss functions.

4.3.3 The listwise loss function for learning to rank

The ListNet Loss

Deriving a loss from the cross-entropy

In order to compare a predicted score vector $\hat{y}_i = (\hat{y}_i^1, \dots, \hat{y}_i^m)$ with a true score vector $y_i = (y_i^1, \dots, y_i^m)$, the objective is to define two probability measures derived from y_i and $\hat{y}_i$, denoted respectively $\mathbb{P}_{y_i}$ and $\mathbb{P}_{\hat{y}_i}$. We can then define $l(\hat{y}_i, y_i)$ in 4.3.1 as the cross-entropy between $\mathbb{P}_{y_i}$ and $\mathbb{P}_{\hat{y}_i}$.

$$l(\hat{y}_i, y_i) = -\sum_{\omega\in\Omega} \mathbb{P}_{y_i}(\omega) \log\left(\mathbb{P}_{\hat{y}_i}(\omega)\right)$$

Preliminary definitions

Consider $n \in \mathbb{N}^*$. In order to define the ListNet loss function, we need to define a probability measure on the space of permutations of $\{1, \dots, n\}$, denoted $\mathbb{S}_n$.

To that end, we need to define first the following spaces :

Definition 4.3.1

For all $i_1 \in \{1, \dots, n\}$, we define $\mathbb{S}_n^{i_1}$ as follows :

$$\mathbb{S}_n^{i_1} := \{\sigma \in \mathbb{S}_n \mid \sigma(1) = i_1\}$$

For all $(i_1, \dots, i_j) \in \{1, \dots, n\}^j$, We define recursively $\mathbb{S}_n^{i_1,\dots,i_j}$ as follows :

$$\mathbb{S}_n^{i_1,\dots,i_j} := \{\sigma \in \mathbb{S}_n^{i_1,\dots,i_{j-1}} \mid \sigma(j) = i_j\}$$

As a convention, when $j = 0$:

$$\mathbb{S}_n^{i_1,\dots,i_j} := \mathbb{S}_n$$

We have the following Lemma :

Lemma 4.3.1

Given $s = (s_1, \dots, s_n) \in \mathbb{R}^n$ a list of scores and $\phi(.)$ an increasing and positive function, for all $j \in \{1, \dots, n\}$, for all $\sigma \in \mathbb{S}_n^{i_1,\dots,i_{j-1}}$, the sum $\sum_{k=j}^{n} \phi(s_{\sigma(k)})$ does not depend on σ.
Furthermore, there exists a function Ψ_ϕ such that :

$$\sum_{k=j}^{n} \phi(s_{\sigma(k)}) = \Psi_\phi(s_{i_1}, \dots, s_{i_{j-1}})$$

If $j = 1$, for all $\sigma \in \mathbb{S}_n^{i_1,\dots,i_{j-1}}$ (i.e, $\sigma \in \mathbb{S}_n$), there exists a constant which does not depend on σ such that :

$$\sum_{k=1}^{n} \phi(s_{\sigma(k)}) = \alpha_\phi$$

Démonstration : The proof of the lemma is straightforward, it relies on the fact that the sum $\sum_{k=1}^{n} \phi(s_{\sigma(k)})$ does not depend on σ. Indeed,

$$\forall \sigma \in \mathbb{S}_n \quad \sum_{k=1}^{n} \phi(s_{\sigma(k)}) = \sum_{k=1}^{n} \phi(s_k) = \alpha_\phi$$

Consequently, for all $\sigma \in \mathbb{S}_n^{i_1,\dots,i_{j-1}}$:

$$\begin{aligned}
\sum_{k=j}^{n} \phi(s_{\sigma(k)}) &= \sum_{k=1}^{n} \phi(s_{\sigma(k)}) - \sum_{k=1}^{j-1} \phi(s_{\sigma(k)}) \\
&= \alpha_\phi - \sum_{k=1}^{j-1} \phi(s_{i_k}) \quad (\text{since} \quad \forall \sigma \in \mathbb{S}_n^{i_1,\dots,i_{j-1}} \quad \forall k \in \{1, \dots, j-1\} \quad \sigma(k) = i_k) \\
&:= \Psi_\phi(s_{i_1}, \dots, s_{i_{j-1}})
\end{aligned}$$

■

Probability measure on the space of permutations

Let us introduce a probability measure on the space of permutations of $\{1, \dots, n\}$.

Theorem 4.3.1 (*Probability measure on the Space of Permutations*)

Let $s = (s_1, \dots, s_n) \in \mathbb{R}^n$ be a list of scores and $\phi(.)$ a positive function. We define a probability measure $\mathbb{P}_s^\phi(.)$ on the the space $\mathbb{S}_n$ of the permutations of $\{1, \dots, n\}$ as follows :

$$\forall \sigma \in \mathbb{S}_n \quad \mathbb{P}_s^\phi(\sigma) := \prod_{j=1}^{n} \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})}$$

Démonstration :

$$\begin{aligned}
\sum_{\sigma \in \mathbb{S}_n} \mathbb{P}_s^\phi(\sigma) &= \sum_{\sigma \in \mathbb{S}_n} \prod_{j=1}^{n} \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \\
&= \sum_{i_1=1}^{n} \sum_{\sigma \in \mathbb{S}_n^{i_1}} \prod_{j=1}^{n} \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \\
&= \sum_{i_1=1}^{n} \sum_{\sigma \in \mathbb{S}_n^{i_1}} \frac{\phi(s_{i_1})}{\sum_{k=1}^{n} \phi(s_{\sigma(k)})} \cdot \frac{\phi(s_{\sigma(2)})}{\sum_{k=2}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(n)})}{\sum_{k=n}^{n} \phi(s_{\sigma(k)})} \\
&= \sum_{i_1=1}^{n} \frac{\phi(s_{i_1})}{\alpha_\phi} \cdot \sum_{\sigma \in \mathbb{S}_n^{i_1}} \frac{\phi(s_{\sigma(2)})}{\sum_{k=2}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(n)})}{\sum_{k=n}^{n} \phi(s_{\sigma(k)})} \\
&= \sum_{i_1=1}^{n} \frac{\phi(s_{i_1})}{\alpha_\phi} \cdot \sum_{\substack{i_2=1 \\ i_2 \neq i_1}}^{n} \sum_{\sigma \in \mathbb{S}_n^{i_1, i_2}} \frac{\phi(s_{i_2})}{\sum_{k=2}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(n)})}{\sum_{k=n}^{n} \phi(s_{\sigma(k)})} \\
&= \sum_{i_1=1}^{n} \frac{\phi(s_{i_1})}{\alpha_\phi} \cdot \sum_{\substack{i_2=1 \\ i_2 \neq i_1}}^{n} \frac{\phi(s_{i_2})}{\Psi_\phi(s_{i_1})} \cdot \sum_{\sigma \in \mathbb{S}_n^{i_1, i_2}} \frac{\phi(s_{\sigma(3)})}{\sum_{k=3}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(n)})}{\sum_{k=n}^{n} \phi(s_{\sigma(k)})} \\
&= \sum_{i_1=1}^{n} \frac{\phi(s_{i_1})}{\alpha_\phi} \cdots \sum_{\substack{i_j=1 \\ i_j \neq i_k \\ \forall k<j}}^{n} \frac{\phi(s_{i_j})}{\Psi_\phi(s_{i_1}, \cdots, s_{i_{j-1}})} \cdots \sum_{\substack{i_n=1 \\ i_n \neq i_k \\ \forall k<n}}^{n} \frac{\phi(s_{i_n})}{\Psi_\phi(s_{i_1}, \cdots, s_{i_{n-1}})}
\end{aligned}$$

Additionally, for all pairwise distinct $i_1, \dots, i_{j-1} \in \{1, \dots, n\}$, we have :

$$\big\{ i_j \in \{1, \dots, n\} \mid \forall k \in \{1, \dots, j-1\} \quad i_j \neq i_k \big\} = \{1, \dots, n\} \backslash \{i_1, \dots, i_{j-1}\}$$

Consequently,

$$\begin{aligned}
\sum_{\substack{i_j=1 \\ i_j \neq i_k \\ \forall k<j}}^{n} \frac{\phi(s_{i_j})}{\Psi_\phi(s_{i_1}, \cdots, s_{i_{j-1}})} &= \frac{1}{\Psi_\phi(s_{i_1}, \cdots, s_{i_{j-1}})} \sum_{\substack{i_j=1 \\ i_j \neq i_k \\ \forall k<j}}^{n} \phi(s_{i_j}) \\
&= \frac{1}{\Psi_\phi(s_{i_1}, \cdots, s_{i_{j-1}})} \sum_{k=j}^{n} \phi(s_{\tilde{\sigma}(k)}) \quad \text{(where } \tilde{\sigma} \text{ is chosen in } \mathbb{S}_n^{i_1,\ldots,i_{j-1}}\text{)} \\
&= \frac{1}{\Psi_\phi(s_{i_1}, \cdots, s_{i_{j-1}})} \Psi_\phi(s_{i_1}, \cdots, s_{i_{j-1}}) \\
&= 1
\end{aligned}$$

Which gives :

$$\sum_{\sigma \in \mathbb{S}_n} \mathbb{P}_s^\phi(\sigma) = 1$$

And as by definition for $\forall \sigma \in \mathbb{S}_n,\ \mathbb{P}_s^\phi(\sigma) \geq 0$

Therefore, $\mathbb{P}_s^\phi(.)$ defines a probability measure on the space $\mathbb{S}_n$. ■

From the previous probability measure, we can derive the proposition 4.3.1.

Proposition 4.3.1

Suppose ϕ is an increasing function. Consider $s = (s_1, \ldots, s_n) \in \mathbb{R}^n$ and two permutations $\sigma, \sigma' \in \mathbb{S}_n$ such that

- $\exists\, p, q \in \{1, \ldots, n\}$ *s.t* $p < q,\ \sigma(p) = \sigma'(q),\ \sigma(q) = \sigma'(p)$,
- $\forall r \in \{1, \ldots, n\} \backslash \{p, q\}\ \sigma(r) = \sigma'(r)$
- $s_{\sigma(p)} \geq s_{\sigma(q)}$

Then,

$$\mathbb{P}_s^\phi(\sigma) \geq \mathbb{P}_s^\phi(\sigma')$$

Démonstration : From the theorem 4.3.1, we get

$$\mathbb{P}_s^\phi(\sigma) = \prod_{j=1}^{n} \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \quad \text{and} \quad \mathbb{P}_s^\phi(\sigma') = \prod_{j=1}^{n} \frac{\phi(s_{\sigma'(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma'(k)})} \tag{4.3.2}$$

Because of hypothesis, in order to prove that $\mathbb{P}_s^{\phi}(\sigma) \geq \mathbb{P}_s^{\phi}(\sigma')$, it suffices to prove that

$$\prod_{j=p}^{q} \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \geq \prod_{j=p}^{q} \frac{\phi(s_{\sigma'(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma'(k)})}$$

By noticing, that $\prod_{j=p}^{q} \phi(s_{\sigma(j)}) = \prod_{j=p}^{q} \phi(s_{\sigma'(j)})$, it suffices to prove that :

$$\prod_{j=p}^{q} \frac{1}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \geq \prod_{j=p}^{q} \frac{1}{\sum_{k=j}^{n} \phi(s_{\sigma'(k)})} \tag{4.3.3}$$

For $p < j \leq q$, we have :

$$\forall k \in \{j, \ldots, n\} \backslash \{q\} \ \phi(s_{\sigma(k)}) = \phi(s_{\sigma'(k)}) \tag{4.3.4}$$

And, due to the increasing property of $\phi(.)$, we have

$$\phi(s_{\sigma(q)}) \leq \phi(s_{\sigma(p)}) = \phi(s_{\sigma'(q)}) \tag{4.3.5}$$

Thus,

$$\sum_{k=j}^{n} \phi(s_{\sigma(k)}) \leq \sum_{k=j}^{n} \phi(s_{\sigma'(k)}) \tag{4.3.6}$$

Consequently,

$$\frac{1}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \geq \frac{1}{\sum_{k=j}^{n} \phi(s_{\sigma'(k)})} \tag{4.3.7}$$

For $j = p$, we notice that

$$\sum_{k=j}^{n} \phi(s_{\sigma(k)}) = \sum_{k=j}^{n} \phi(s_{\sigma'(k)}) \tag{4.3.8}$$

Finally, we get (4.3.3) and consequently $\mathbb{P}_s^{\phi}(\sigma) \geq \mathbb{P}_s^{\phi}(\sigma')$ ■

So, if we alternate the positions of two objects, all other things remaining equal, one with a higher score

than the other, we obtain a ranking vector with a lower permutation probability.

We can conclude a way of computing the loss function from the predicted score vector $\hat{y}_i \in \mathbb{R}^m$ and the true score vector vector $y_i \in \mathbb{R}^m$. Indeed, it can be derived from the probability distributions $P^{\phi}_{y_i}$ and $P^{\phi}_{\hat{y}_i}$ as follows :

$$l(\hat{y}_i, y_i) = -\sum_{\sigma \in \mathbb{S}_n} \mathbb{P}^{\phi}_{y_i}(\sigma) \log\left(\mathbb{P}^{\phi}_{\hat{y}_i}(\sigma)\right) \tag{4.3.9}$$

Since $|\mathbb{S}_n| = n!$, the computation can be intractable. (Z. Cao et al., 2007) suggests using the top one probability measure to solve this problem.

The Top one probability measure

Consider $n \in \mathbb{N}^*$. We define the Top one probability measure on $\{1, \ldots, n\}$ as follows :

Definition 4.3.2

Given $s = (s_1, \ldots, s_n) \in \mathbb{R}^n$ a list of scores and $\phi(.)$ an increasing and positive function, The top one probability measure is defined as follows :

$$\forall i \in \{1, \ldots, n\} \; \mathbb{P}_s^{\phi \text{ Top-One}}(i) = \sum_{\substack{\sigma \in \mathbb{S}_n \\ \sigma(1)=i}} \mathbb{P}^{\phi}_s(\sigma)$$

Given $s = (s_1, \ldots, s_n) \in \mathbb{R}^n$, the top one probability of an object i represents the probability that it will rank first. It equals to the sum of the permutation probabilities for all the permutations where the object i is ranked first.

The Top One probability measure solves the $n!$ complexity problem. Indeed, we have the following theorem.

Theorem 4.3.2

For the top one probability $\mathbb{P}_s^{\phi \text{ Top-One}}(i)$, given the vector of score $s = (s_1, \ldots, s_n)$ we have :

$$\mathbb{P}_s^{\phi \text{ Top-One}}(i) = \frac{\phi(s_i)}{\sum_{k=1}^{n} \phi(s_k)}$$

Démonstration :

$$
\begin{aligned}
\mathbb{P}_s^{\phi\text{Top-One}}(i) &= \sum_{\substack{\sigma\in\mathbb{S}_n \\ \sigma(1)=i}} \mathbb{P}_s^{\phi}(\sigma) \\
&= \sum_{\sigma\in\mathbb{S}_n^i} \mathbb{P}_s^{\phi}(\sigma) \\
&= \sum_{\sigma\in\mathbb{S}_n^i} \prod_{j=1}^{n} \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \\
&= \sum_{\sigma\in\mathbb{S}_n^i} \frac{\phi(s_i)}{\sum_{k=1}^{n} \phi(s_{\sigma(k)})} \cdot \frac{\phi(s_{\sigma(2)})}{\sum_{k=2}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(n)})}{\sum_{k=n}^{n} \phi(s_{\sigma(k)})} \\
&= \frac{\phi(s_i)}{\sum_{k=1}^{n} \phi(s_{\sigma(k)})} \underbrace{\sum_{\sigma\in\mathbb{S}_n^i} \frac{\phi(s_{\sigma(2)})}{\sum_{k=2}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(j)})}{\sum_{k=j}^{n} \phi(s_{\sigma(k)})} \cdots \frac{\phi(s_{\sigma(n)})}{\sum_{k=n}^{n} \phi(s_{\sigma(k)})}}_{=1 \quad \text{as shown in Theorem 2.1}} \\
&= \frac{\phi(s_i)}{\sum_{k=1}^{n} \phi(s_{\sigma(k)})} \\
&= \frac{\phi(s_i)}{\sum_{k=1}^{n} \phi(s_k)}
\end{aligned}
$$

■

In addition to solving the complexity problem, the top one probability satisfies the following proposition :

Proposition 4.3.2

Given any two objects i and j such that $s_i \geq s_j$, and suppose that $\phi(.)$ is an increasing function, then

$$\mathbb{P}_s^{\phi\,\textit{Top-One}}(i) \geq \mathbb{P}_s^{\phi\,\textit{Top-One}}(j)$$

Démonstration : The proof is straightforward. Because $s_i \geq s_j$ and increasing behavior of $\phi(.)$, we have : $\phi(s_i) \geq \phi(s_j)$ and then $\mathbb{P}_s^{\phi\text{Top-One}}(i) \geq \mathbb{P}_s^{\phi\text{Top-One}}(j)$ ■

In the following, we choose $\phi(.)$ to be the exponential function.

Therefore, assuming that $\phi(x) = \exp(x)$, we refer to $\mathbb{P}_s^{\phi\text{Top-One}}(.)$ by $\mathbb{P}_s^{\text{Top-One}}(.)$.

As shown in (Z. Cao et al., 2007), we can now derive the loss function from the predicted score vector $\hat{y}_i \in \mathbb{R}^m$ and the true score vector vector $y_i \in \mathbb{R}^m$ as follows :

$$l(\hat{y}_i, y_i) = -\sum_{j=1}^{m} \mathbb{P}_{y_i}^{\text{Top-One}}(j) \log\left(\mathbb{P}_{\hat{y}_i}^{\text{Top-One}}(j)\right) \tag{4.3.10}$$

The RankCosine loss function

The RankCosine Loss has been introduced in (Qin et al., 2008). It is defined as follows :

$$\begin{aligned} l(\hat{y}_i, y_i) &= \frac{1}{2}\big(1 - \cos(y_i, \hat{y}_i)\big) \\ &= \frac{1}{2}\left(1 - \frac{y_i^T \hat{y}_i}{\|y_i\|_2 \|\hat{y}_i\|_2}\right) \end{aligned}$$

where

$$x \in \mathbb{R}^n,\ \|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

The desirable properties of a loss function

The following two propositions describe the properties of a *desirable* loss function.

Proposition 4.3.3

Suppose there exists a score vector $y_i = (y_i^1, \ldots, y_i^m)$ *such that* $y_i^p \geq y_i^q$ *where* $p, q \in \{1, \ldots, m\},\ p > q$ *and two prediction score vectors* $\hat{y}_i = (\hat{y}_i^1, \ldots, \hat{y}_i^m) = f_{\theta_1}(x_i^1, \ldots, x_i^m)$ *and* $\tilde{y}_i = (\tilde{y}_i^1, \ldots, \tilde{y}_i^m) = g_{\theta_2}(x_i^1, \ldots, x_i^m)$ *such that :*

- $\forall k \in \{1, \ldots, m\} \backslash \{p, q\}\ \hat{y}_i^k = \tilde{y}_i^k$
- $\hat{y}_i^p = \tilde{y}_i^q$
- $\hat{y}_i^q = \tilde{y}_i^p$
- $\hat{y}_i^p \leq \hat{y}_i^q$

Then, a "desirable" loss function l *verifies the inequality :* $l(\hat{y}_i, y_i) \geq l(\tilde{y}_i, y_i)$

Proposition 4.3.3 indicates that for any ranking function, if we swap a position in the prediction vector with the higher score with the feature lowest score, then a "desirable" loss function is higher. If we exchange two well-positioned features, the penalty is greater.

Proposition 4.3.4

A "desirable" loss function l *is upper bounded, i.e :*

$$\exists C > 0 \, \forall i \in \{1, \dots, N\} \; 0 \leq l(\hat{y}_i, y_i) \leq C$$

According to the proposal Proposition 4.3.4, a "desirable" Loss function would not be biased by difficult queries. The training process would otherwise be dominated by queries with very large losses.

The ListNet loss function We have the following propositions :

Proposition 4.3.5

The ListNet Loss Function verifies the Proposition 4.3.3.

Démonstration : Consider the true score vector $y_i = (y_i^1, \dots, y_i^m) \in \mathbb{R}^m$ and two prediction score vectors $\hat{y}_i = (\hat{y}_i^1, \dots, \hat{y}_i^m) = f_{\theta_1}(x_i^1, \dots, x_i^m)$ and $\tilde{y}_i = (\tilde{y}_i^1, \dots, \tilde{y}_i^m) = g_{\theta_2}(x_i^1, \dots, x_i^m)$ such that :

— $\forall k \in \{1, \dots, m\} \backslash \{p, q\} \; \hat{y}_i^k = \tilde{y}_i^k$

— $\hat{y}_i^p = \tilde{y}_i^q$

— $\hat{y}_i^q = \tilde{y}_i^p$

— $\hat{y}_i^p \leq \hat{y}_i^q$

We have :

$$\begin{aligned}
l(\hat{y}_i, y_i) - l(\tilde{y}_i, y_i) &= -\sum_{j=1}^{m} \mathbb{P}_{y_i}^{\text{Top-One}}(j) \log\left(\mathbb{P}_{\hat{y}_i}^{\text{Top-One}}(j)\right) + \sum_{j=1}^{m} \mathbb{P}_{y_i}^{\text{Top-One}}(j) \log\left(\mathbb{P}_{\tilde{y}_i}^{\text{Top-One}}(j)\right) \\
&= \mathbb{P}_{y_i}^{\text{Top-One}}(p) \log\left(\mathbb{P}_{\tilde{y}_i}^{\text{Top-One}}(p)\right) + \mathbb{P}_{y_i}^{\text{Top-One}}(q) \log\left(\mathbb{P}_{\tilde{y}_i}^{\text{Top-One}}(q)\right) \\
&\quad - \mathbb{P}_{y_i}^{\text{Top-One}}(p) \log\left(\mathbb{P}_{\hat{y}_i}^{\text{Top-One}}(p)\right) - \mathbb{P}_{y_i}^{\text{Top-One}}(q) \log\left(\mathbb{P}_{\hat{y}_i}^{\text{Top-One}}(q)\right) \\
&= \mathbb{P}_{y_i}^{\text{Top-One}}(p) \log\left(\frac{\mathbb{P}_{\tilde{y}_i}^{\text{Top-One}}(p)}{\mathbb{P}_{\hat{y}_i}^{\text{Top-One}}(p)}\right) + \mathbb{P}_{y_i}^{\text{Top-One}}(q) \log\left(\frac{\mathbb{P}_{\tilde{y}_i}^{\text{Top-One}}(q)}{\mathbb{P}_{\hat{y}_i}^{\text{Top-One}}(q)}\right)
\end{aligned}$$

As $y_i^p \geq y_i^q$, we have

$$\mathbb{P}_{y_i}^{\text{Top-One}}(p) = \frac{\exp(y_i^p)}{\sum_{j=1}^{m} \exp(y_i^j)} \geq \frac{\exp(y_i^q)}{\sum_{j=1}^{m} \exp(y_i^j)} = \mathbb{P}_{y_i}^{\text{Top-One}}(q)$$

Which gives :

$$l(\hat{y}_i, y_i) - l(\tilde{y}_i, y_i) \geq \mathbb{P}_{y_i}^{\text{Top-One}}(q) \left[\log\left(\frac{\mathbb{P}_{\tilde{y}_i}^{\text{Top-One}}(p)}{\mathbb{P}_{\hat{y}_i}^{\text{Top-One}}(p)}\right) + \log\left(\frac{\mathbb{P}_{\tilde{y}_i}^{\text{Top-One}}(q)}{\mathbb{P}_{\hat{y}_i}^{\text{Top-One}}(q)}\right)\right]$$

And since $\tilde{y}_i^p = \hat{y}_i^q$ and $\{\tilde{y}_i^1, \dots, \tilde{y}_i^m\} = \{\hat{y}_i^1, \dots, \hat{y}_i^m\}$, we have :

$$\begin{aligned}\mathbb{P}^{\text{Top-One}}_{\hat{y}_i}(q) &= \frac{\exp(\hat{y}_i^q)}{\sum_{j=1}^{m}\exp(\hat{y}_i^j)} \\ &= \frac{\exp(\tilde{y}_i^p)}{\sum_{j=1}^{m}\exp(\tilde{y}_i^j)} \\ &= \mathbb{P}^{\text{Top-One}}_{\tilde{y}_i}(p)\end{aligned}$$

Similarly, we have

$$\mathbb{P}^{\text{Top-One}}_{\hat{y}_i}(p) = \mathbb{P}^{\text{Top-One}}_{\tilde{y}_i}(q)$$

Consequently,

$$l(\hat{y}_i, y_i) - l(\tilde{y}_i, y_i) \geq \mathbb{P}^{\text{Top-One}}_{y_i}(q) \underbrace{\left[\log\left(\frac{\mathbb{P}^{\text{Top-One}}_{\tilde{y}_i}(p)}{\mathbb{P}^{\text{Top-One}}_{\tilde{y}_i}(q)}\right) + \log\left(\frac{\mathbb{P}^{\text{Top-One}}_{\tilde{y}_i}(q)}{\mathbb{P}^{\text{Top-One}}_{\tilde{y}_i}(p)}\right)\right]}_{=0}$$

Finally,

$$l(\hat{y}_i, y_i) - l(\tilde{y}_i, y_i) \geq 0$$

■

Proposition 4.3.6

The ListNet Loss Function does not verify the Proposition 4.3.4.

Démonstration : Since the cross entropy cannot be bounded, a lot of examples can be found to prove the point, here is one of them :

Consider the following vectors $y_i = (y_i^1, \ldots, y_i^m)$ and $\hat{y}_i = (\hat{y}_i^1, \ldots, \hat{y}_i^m)$ such that :

$$\forall j \in \{1, \ldots, m\}\ y_i^j = \frac{1}{m} = \hat{y}_i^j$$

Therefore,

$$\begin{aligned}l(\hat{y}_i, y_i) &= -\sum_{j=1}^{m}\mathbb{P}^{\text{Top-One}}_{y_i}(j)\log(\mathbb{P}^{\text{Top-One}}_{\hat{y}_i}(j)) \\ &= -\frac{1}{m}m\log\left(\frac{1}{m}\right) \\ &= -\log\left(\frac{1}{m}\right) \xrightarrow[m\to+\infty]{} +\infty\end{aligned}$$

■

The RankCosine loss function

Proposition 4.3.7

The RankCosine Loss function verifies the Proposition 4.3.3.

Démonstration : Consider the true score vector $y_i = (y_i^1, \dots, y_i^m) \in \mathbb{R}^m$ and two prediction score vectors $\hat{y}_i = (\hat{y}_i^1, \dots, \hat{y}_i^m) = f_{\theta_1}(x_i^1, \dots, x_i^m)$ and $\tilde{y}_i = (\tilde{y}_i^1, \dots, \tilde{y}_i^m) = g_{\theta_2}(x_i^1, \dots, x_i^m)$ such that :

— $\forall k \in \{1, \dots, m\} \backslash \{p, q\}$ $\hat{y}_i^k = \tilde{y}_i^k$

— $\hat{y}_i^p = \tilde{y}_i^q$

— $\hat{y}_i^q = \tilde{y}_i^p$

— $\hat{y}_i^p \leq \hat{y}_i^q$

We have :

$$\begin{aligned} l(\hat{y}_i, y_i) - l(\tilde{y}_i, y_i) &= \frac{1}{2}\left(1 - \frac{y_i^T \hat{y}_i}{\|y_i\|_2 \|\hat{y}_i\|_2}\right) - \frac{1}{2}\left(1 - \frac{y_i^T \tilde{y}_i}{\|y_i\|_2 \|\tilde{y}_i\|_2}\right) \\ &= -\frac{y_i^T}{2\|y_i\|_2}\left(\frac{\hat{y}_i}{\|\hat{y}_i\|_2} - \frac{\tilde{y}_i}{\|\tilde{y}_i\|_2}\right) \end{aligned}$$

As $\{\tilde{y}_i^1, \dots, \tilde{y}_i^m\} = \{\hat{y}_i^1, \dots, \hat{y}_i^m\}$, we have :

$$\|\tilde{y}_i\|_2 = \|\hat{y}_i\|_2$$

Therefore,

$$\begin{aligned} l(\hat{y}_i, y_i) - l(\tilde{y}_i, y_i) &= -\frac{y_i^T}{2\|y_i\|_2}\left(\frac{\hat{y}_i}{\|\hat{y}_i\|_2} - \frac{\tilde{y}_i}{\|\tilde{y}_i\|_2}\right) \\ &= -\frac{y_i^T}{2\|y_i\|_2 \|\tilde{y}_i\|_2}\left(\hat{y}_i - \tilde{y}_i\right) \end{aligned}$$

On the other hand :

$$\hat{y}_i - \tilde{y}_i = (0, \dots, 0, \hat{y}_i^q - \tilde{y}_i^q, 0, \dots, 0, \hat{y}_i^p - \tilde{y}_i^p, 0 \dots, 0)$$

And

$$\hat{y}_i^q = \tilde{y}_i^p \quad \text{and} \quad \hat{y}_i^p = \tilde{y}_i^q$$

Consequently,

$$
\begin{aligned}
l(\hat{y}_i, y_i) - l(\tilde{y}_i, y_i) &= \frac{1}{2\|y_i\|_2\|\tilde{y}_i\|_2}\left(-y_i^q(\hat{y}_i^q - \tilde{y}_i^q) - y_i^p(\hat{y}_i^p - \tilde{y}_i^p)\right) \\
&= \frac{1}{2\|y_i\|_2\|\tilde{y}_i\|_2}\left(-y_i^q(\tilde{y}_i^p - \tilde{y}_i^q) - y_i^p(\tilde{y}_i^q - \tilde{y}_i^p)\right) \\
&= \frac{1}{2\|y_i\|_2\|\tilde{y}_i\|_2}\underbrace{(\tilde{y}_i^p - \tilde{y}_i^q)}_{\geq 0}\underbrace{(-y_i^q + y_i^p)}_{\geq 0} \\
&\geq 0
\end{aligned}
$$

■

Proposition 4.3.8

The RankCosine Loss function verifies the Proposition 4.3.4

Démonstration : By definition, $C = 1$ is an upper bound of the RankCosine Loss. ■

4.4 The Ranking System

4.4.1 Introducing the problem

We consider the tree structure represented in figure 4.2. An index is composed of N_S sectors, N_i industries and N stocks.

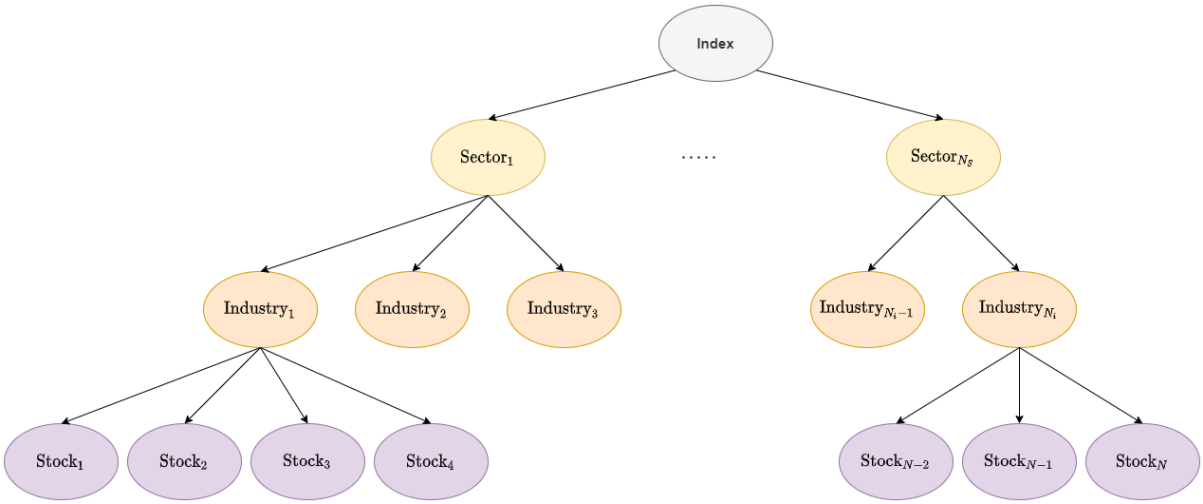


FIGURE 4.2 – Ranking Universe

The objective of the Ranking System is to predict at each time step, the ranking over the next period for each group of children associated with a parent node.

To that end, the model represented in figure 4.3 consists in two steps :

— **Step 1 :** Getting the sequence of skill vectors associated with a group of children. This step is presented in section 4.4.2

— **Step 2 :** Learning the dynamics of the skill vectors using a sequential model with attention me-

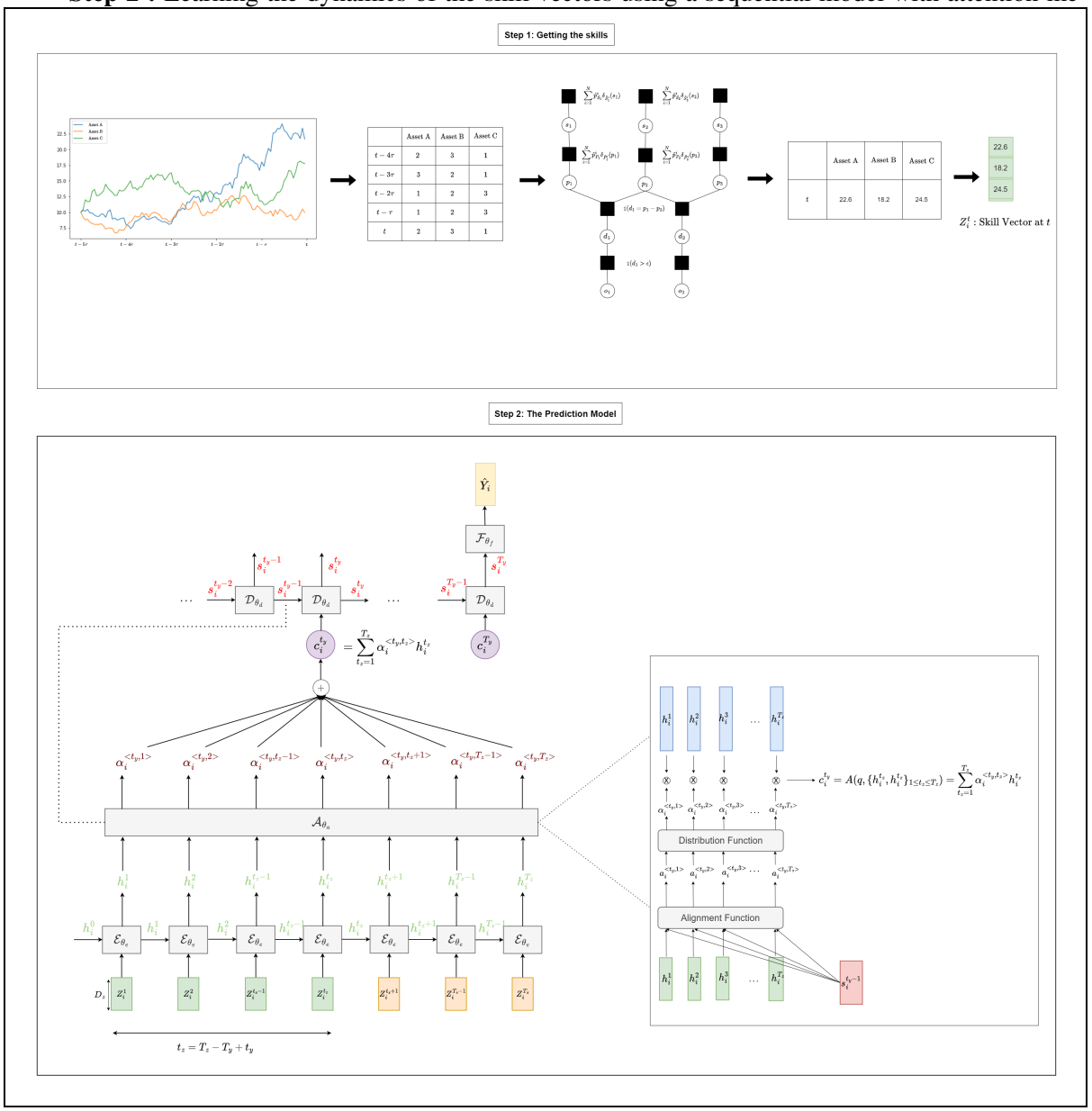


FIGURE 4.3 – The Ranking System

4.4.2 Bayesian Approach for learning Asset skills

Introduction

In a financial world largely guided by indices, outperforming them has become a goal within the financial community. Many traditional techniques rely on equal weighting, maximizing diversification using certain filters, or rules, such as volume, size, indicator etc. Contest Theory enables us to have a more detailed understanding of all the constituents of an index and therefore to select a subset of it based on a

more refined filtering process.

Contest Theory has been applied in the context of video games to rank players their past results. The TrueSkill Algorithm (Herbrich et al., 2006) developed by Microsoft allows us to infer skill players from match result observations. We describe TrueSkill Factor Graph and an inference algorithm called Optimal Quantized Belief Propagation presented in (Chichportich et al., 2020) that we use to infer the skill of financial assets.

TrueSkill

For more information on TrueSkill, please see (Herbrich et al., 2006). The main purpose of TrueSkill is to identify skill players through match results. In our case, the players are the children of the same groups. We fix a time t. The framework used is as follows :

— Each players has prior skill $S_i^t \sim \mathcal{N}(\mu_i, \sigma_0^2)$ independent from others players

— We add a noise to prior with a performance variable : $P_i^t | S_i^t \sim \mathcal{N}(S_i^t, \beta^2)$

— We build the difference between performance variables : $d_i^t | P_j^t, P_k^t \sim P_j^t - P_k^t$

— We observe the difference : $I | d_i^t \; I(d_i^t) = \begin{cases} 1 & \text{if } d_i^t < -\epsilon \\ 0 & \text{if } |d_i^t| < \epsilon \\ -1 & \text{if } d_i^t > \epsilon \end{cases}$

Therefore, from priors skills and match results, (Herbrich et al., 2006) applies Expectation Propagation inference algorithm as described in (Minka, 2001) to update the skill iteratively.

The following section describes another algorithm of inference that we used for iteratively updating the skill.

Inference of skill

The following section outlines the general inference algorithms we use in the rest of the paper .There is a special notation used throughout this section. We can determine assets skill using Bayesian inference by updating the probability distribution as new information becomes available. Mathematically, Bayesian inference involve the following setting. We fix two random vectors $X = (X_1, \cdots, X_p)$ and $\Theta = (\Theta_1, \cdots, \Theta_p)$ defined on $(\Omega, \mathcal{A}, \mathbb{P})$ that represent respectively the observations and the parameters. We assume that Θ admits a density, the *prior*, with respect to Lebesgue measure λ_p on $\mathbb{R}^p$ that we note $p_\Theta(.)$. The prior is then updated thanks to Bayes Theorem by conditioning over the observations X leading to the *posterior* distribution $\pi_{\Theta|X}(.)$. We denote by $l(.)$ the *likelihood* distribution, which represent the random variable $X|\Theta$. We suppose also that $l(\theta, x)$ admits a density with respect to Lebesgue

measure λ_p. Then, by Bayes Theorem :

$$\begin{aligned}\pi_{\Theta|X=x}(\theta) &= \frac{l(\theta,x)p_\Theta(\theta)}{p_X(x)} \\ &= \frac{l(\theta,x)p_\Theta(\theta)}{\int_{\mathbb{R}^p} l(\theta,x)p_\Theta(\theta)d\lambda_p(\theta)}\end{aligned}$$

Most of the time, the posterior cannot be computed. In this section, we describe the Optimal Quantized Belief Propagation method for approximating inference. Based on Belief Propagation and Optimal Quantization theory, We describe the algorithm and refer to (Chichportich et al., 2020) for more informations.

Belief Propagation We briefly recall the Belief Propagation algorithm. Let us first describe BP using the formalism of factor graphs. In this part, we outline the original BP algorithm described in (Mezard & Montanari, 2009), and we follow that description.

Probability distribution. We fix p finite sets $\mathcal{T}_1, \dots, \mathcal{T}_p$ and a probability function p on $\mathcal{T} := \mathcal{T}_1 \times \cdots \times \mathcal{T}_p$. That is $p(\theta) \geq 0$ for all $\theta = (\theta_1, \dots, \theta_p) \in \mathcal{T}$ and $\sum_{\theta \in \mathcal{T}} p(\theta) = 1$. We suppose that the probability function $p(.)$ admits the following decomposition

$$p(\theta) = \frac{1}{\mathcal{Z}} \prod_{a=1}^{m} \psi_a(\theta_{\partial a}) \qquad \text{for } \theta \in \mathcal{T} ,$$

where ∂a is a given subset of $\{1, \dots, p\}$ and $\theta_{\partial a} := (\theta_i)_{i \in \partial a}$ and $\mathcal{Z}$ a normalization constant. In order to compute marginal functions of $p(.)$, a possible naive approach consists in computing naively

$$p_i(\theta_i) = \sum_{\theta_1} \cdots \sum_{\theta_{i-1}} \sum_{\theta_{i+1}} \cdots \sum_{\theta_p} p(\theta_1, \dots, \theta_p) .$$

Unfortunately, the complexity of this “brute force" approach is $O(N^{p-1})$ whenever each set contains N elements. On the opposite, Belief Propagation intends to provide an efficient an tractable algorithm for the computation of the marginal functions $(p_i)_i$.

Factor Graph. In order to describe Belief Propagation, let us first define the corresponding factor graph Γ. The function node set F is given by $F = \{a \; ; \; a = 1, \dots, m\}$ and the variable node set V by $V = \{i \; ; \; i = 1, \dots, p\}$. The factor graph Γ is a a subset of $V \times F$ composed by the couples (i, a) such that $i \in \partial a$. In the sequel, we suppose that our factor graph Γ is a tree with diameter t^* (see (Mezard & Montanari, 2009)[Part A, Section 3.1.1] for the precise definition).

Message passing. The BP algorithm consists in passing iteratively messages between variable nodes i and function nodes a such that $(i,a) \in \Gamma$. For $i \in V$, we define the set ∂i by

$$\partial i \;=\; \{a \in F \,:\, i \in \partial a\}\,.$$

At the t-th iteration of the BP algorithm, there are, for each edge $(i,a) \in \Gamma$, two messages $m^t_{i\to a}$ and $m^t_{a\to i}$. The messages are first defined by an initial condition $m^0_{i\to a}$ and $m^0_{a\to i}$ that are probability measures on $\mathcal{T}_i$ for $i = 1,\dots,p$. Then, the messages are updated according to the following equations

$$m^{t+1}_{i\to a}(\theta_i) \;=\; \frac{1}{C^t_{i\to a}} \prod_{b\in\partial i\setminus a} m^t_{b\to i}(\theta_i)\,, \tag{4.4.11}$$

$$m^{t+1}_{a\to i}(\theta_i) \;=\; \frac{1}{C^t_{a\to i}} \sum_{(\theta_j)_{j\in\partial a\setminus i}} \psi_a(\theta_{\partial a}) \prod_{j\in\partial a\setminus i} m^t_{j\to a}(\theta_j)\,, \tag{4.4.12}$$

where the constants $C^t_{i\to a}$ and $C^t_{a\to i}$ allow messages to remain probability measures and are given by :

$$\begin{aligned} C^t_{i\to a} &:= \sum_{\theta_i\in\mathcal{T}_i} \prod_{b\in\partial j\setminus a} m^t_{b\to j}(\theta_i)\,,\\ C^t_{a\to i} &:= \sum_{\theta_i\in\mathcal{T}_i} \sum_{(\theta_j)_{j\in\partial a\setminus i}} \psi_a(\theta_{\partial a}) \prod_{j\in\partial a\setminus i} m^t_{j\to a}(\theta_j)\,. \end{aligned}$$

Marginal approximation. At step t, the approximation of the marginal distributions $(p_i)_i$ are given by $(m^t_i)_i$ where

$$m^t_i(\theta_i) \;\propto\; \prod_{a\in\partial i} m^t_{a\to i}(\theta_i)$$

for all $\theta_i \in \mathcal{T}_i$ and $i = 1,\dots,p$ and $\propto$ denotes equality up a normalization term for the global distribution.

Convergence. Since the factor graph Γ is a tree with diameter t^*, BP converges in at most t^* iterations. We refer to (Mezard & Montanari, 2009)[Part D, Theorem 14.1] for a proof of this result that we report here for better readability.

Theorem 4.4.1 (*BP convergence*)

In Tree Graphical model and for any given initial condition, the sequence of messages defined by (4.4.13)-(4.4.13) *converges after at most t^* iteration :*

$$\begin{aligned} m^t_{i\to a}(\theta_i) &= m^{t^*}_{i\to a}(\theta_i)\\ m^t_{a\to i}(\theta_i) &= m^{t^*}_{a\to i}(\theta_i) \end{aligned}$$

for any $t \geq t^*$. Moreover, the limit provides the exact marginal functions : $(m_i^{t^*})_i$ and $(p_i)_i$ coincide on $\mathcal{T}$.

Optimal Quantization We refer to (Graf & Luschgy, 2000) for a complete description of quantization methods. Let fix a random variable Y defined on $(\Omega, \mathcal{A}, \mathbb{P})$ and valued in $\mathbb{R}^\ell$ and suppose that $Y \in L^2(\Omega, \mathcal{A}, \mathbb{P})$. For a fixed integer N, the goal of the optimal quantization is to find the best approximation in a quadratic sense of the random variable Y by a random variable $\tilde{Y}$ taking at most N values.

Voronoï tesselation. To construct such an approximation, the first step is to divide the space $\mathbb{R}^\ell$ into a partition of size N as in the following definition.

Definition 4.4.1 (*Voronoï partition*)

Being given $y = (y_1, \ldots, y_N) \in (\mathbb{R}^\ell)^N$ a so-called a N-quantizer, a Borel partition $(C_i(y))_{1 \leq i \leq N}$ of $\mathbb{R}^\ell$ is a Voronoï partition induced by y if

$$C_i(y) \quad \subset \quad \left\{ \xi \in \mathbb{R}^\ell \ : \ |\xi - y_i| = \min_{1 \leq j \leq n} |\xi - y_j| \right\}$$

for all $i = 1, \ldots, N$. The Borel sets $C_i(y)$ are called the Voronoi cells of the partition induced by y.

Definition 4.4.2 (*Projection operator*)

Given an N-quantizer $y = (y_1, \ldots, y_N) \in (\mathbb{R}^\ell)^N$ and induced Voronoi tesselation $(C_i(y))_{1 \leq i \leq N}$, we define the associated projection operator

$$\mathit{Proj}_y : \xi \in \mathbb{R}^\ell \mapsto \sum_{i=1}^{N} y_i \mathbb{1}_{C_i(y)}(\xi) \, . \tag{4.4.13}$$

We are now in position to introduce the y-Voronoi quantizer $\hat{Y}^y$ of Y by $\hat{Y}^y := \mathrm{Proj}_y(Y)$. By construction, the quantized random variable $\hat{Y}^y$ only takes values in $\{y_1, \ldots, y_N\}$ and its distribution is given by

$$\mathbb{P}(\hat{Y}^y = y_i) \quad := \quad \mathbb{P}(Y \in C_i(y)) \, , \quad i = 1, \ldots, N \, .$$

Distortion. We can now define the quadratic distortion function D_N^Y at level N by

$$D_N^Y(y) \quad := \quad \mathbb{E}\Big[\min_{1 \leq i \leq N} |Y - y_i|^2 \Big] \ = \ \mathbb{E}\Big[|Y - \hat{Y}^y|^2 \Big] \, . \tag{4.4.14}$$

This metric quantifies the accuracy of the quantizer $\hat{Y}^y$ for approximating Y. We hence naturally aim at finding the optimal N-quantizer $(y_1, \dots, y_N)$ by solving the optimization problem

$$\underline{D_N^Y} \;:=\; \inf_{y\in(\mathbb{R}^\ell)^N} D_N^Y(y)\ .$$

Accuracy. We finally collect the main properties of the quantization algorithm, ensuring the attainability of an optimal quantizer, as well as the speed of convergence of the induced discrepancy $\underline{D_N^Y}$ to 0 when the number of quantization points converges to infinity. We refer the reader to (Zador, 1963) and (Zador, 1982) for a proof of those well known results.

Theorem 4.4.2 (*Quantization properties*)

(i) ***Attainability :*** *For any N, there exists $y^* \in (\mathbb{R}^\ell)^N$ s. t.*

$$D_N^Y(y^*) \;=\; \underline{D_N^Y}\ .$$

(ii) ***Convergence :*** *$\underline{D_N^Y} \to 0$ as $N \to \infty$.*

(iii) ***Zador's Theorem*** *Fix $p \in (0, +\infty)$ and suppose that $Y \in L^{p+\delta}(\Omega, \mathcal{A}, \mathbb{P})$ and $\mathbb{P}_Y(d\xi) = \varphi(\xi)\lambda_\ell(d\xi) + \nu(d\xi)$, where ν is singular with respect to the Lebesgue measure λ_ℓ on $\mathbb{R}^\ell$. Then, there is a constant $J_{p,\ell} \in (0, +\infty)$ such that*

$$\lim_{N\to+\infty} N^{\frac{1}{\ell}}\underline{D_N^Y} \;=\; J_{p,\ell}\Big(\int_{\mathbb{R}^\ell} \varphi^{\frac{l}{p+\ell}}\Big)^{\frac{1}{p}+\frac{1}{\ell}}\ .$$

Optimal Quantized Belief Propagation We described the Optimal Quantized Belief Propagation as presented in (Chichportich et al., 2020). We denote by X the random vector of observations $X = (X_1, \dots, X_n)$ taking values in a finite set $\mathcal{X} = \{x_1, \dots, x_m\}$ and we set $p_X(x) = \mathbb{P}(X = x)$ for $x \in \mathcal{X}^n$.

We suppose that the random vector $\Theta = (\Theta_1, \dots, \Theta_p)$ admits a density w.r.t. Lebesgue measure λ_p on $\mathbb{R}^p$ and we denote by p_Θ this density. We also suppose that the conditional law of X given Θ is of the form

$$\mathbb{P}(X = x|\Theta = \theta) \;=\; \prod_{u=1}^{p} \ell_u(\theta_{\partial u}, x)\ . \tag{4.4.15}$$

The posterior marginal density $\pi_\Theta(\cdot|X=x)$ of Θ given the observations $X=x$ is given by :

$$\pi_\Theta(\theta|X=x) \quad = \quad \frac{p_\Theta(\theta)}{p_X(x)} \prod_{u=1}^{p} \ell_u(\theta_{\partial u}, x) \ .$$

The goal of the algorithm is to provide an approximation for the marginal posterior densities $\pi_{\Theta_i}(\cdot|X=x)$ of Θ_i, $i=1,\dots,p$, given the observations $X=x$.

We now present the details of the OQBP algorithm which is divided in three main steps.

1. **Quantization of the prior.** An integer $N \geq 1$ is fixed and the prior law $\mathbb{P}_\Theta$ is approximated by its N optimal quantized $\hat{\theta} = (\hat{\theta}^1, \dots, \hat{\theta}^N)$. We denote by $(C_i(\hat{\theta}))_{1\leq i\leq N}$ the related Voronoï tessellation, by $\hat{\Theta}$ the quantized random variable and by $p_{\hat{\Theta}}$ the probability measure defined by

$$p_{\hat{\Theta}}(\theta) \quad = \quad \sum_{i=1}^{N} \hat{p}^i_\Theta \delta_{\hat{\theta}^i}(\theta) \ .$$

   where

$$\hat{p}^i_\Theta \quad = \quad \int_{C_i(\hat{\theta})} p_\Theta(\theta) d\lambda_p(\theta)$$

   for $i=1,\dots,N$. The computation of this quantized prior is done using the Lloyd Algorithm 1, see (Pagès, 2018)[Section 5.3.1] for details.

2. **Computation of the posterior using the quantized prior.** In (Chichportich et al., 2020) one's makes the following assumption.

   (H$\mathcal{F}$) The conditional law of X given Θ can be decomposed as follows

$$\mathbb{P}(X=x|\hat{\Theta}=\theta) \quad = \quad \prod_{u=1}^{q} \hat{\ell}_u(\theta_{\partial u}, x) \ . \tag{4.4.16}$$

   Under **(H$\mathcal{F}$)**, the posterior law of $\hat{\Theta}$ given $X=x$ is then given by

$$\pi_{\hat{\Theta}}(\theta|X=x) \quad = \quad \frac{p_{\hat{\Theta}}(\theta)}{p_X(x)} \prod_{u=1}^{q} \hat{\ell}_u(\theta_{\partial u}, x) \ .$$

   The computation of the marginal posterior law of $\hat{\Theta}_i$ given $X=x$ is done using BP by taking $\mathcal{U} = \{0,\dots,q\}$, $\psi_0(\cdot) = p_{\hat{\Theta}}(\cdot)$ and

$$\psi_u(\cdot) = \hat{\ell}_u(\cdot, x)$$

for $u = 1, \dots, q$ and then :

$$\pi_{\hat{\Theta}}(\theta, X = x) \quad = \quad \prod_{u=0}^{q} \psi_u(\theta_{\partial u}) \ .$$

Because the TrueSkill Factor Graph is a tree, we got by Theorem 4.4.1 that BP provides the exact marginals posteriors that we denote by $\pi_{\hat{\Theta}_i}(.|X = x)$. These posteriors are called quantized posteriors.

3. **Transform the quantized posterior into an absolutely continuous law w.r.t. Lebesgue measure.** The quantized posterior has a finite support and is of the form

$$\pi_{\hat{\Theta}_i}(\cdot|X = x) \quad = \quad \sum_{j=1}^{N} \hat{p}_i^{j,x} \delta_{\hat{\theta}_i^j} \ .$$

A compact is fixed $K \subset \mathbb{R}^p$ and the law $\pi_{\hat{\Theta}_i}(\cdot|X = x)$ is transformed into the absolutely continuous law $\bar{\pi}_{\Theta_i}(\cdot|X = x)$ where $d\bar{\pi}_{\Theta_i}(\theta_i|X = x)$ is given by

$$\begin{aligned} \sum_{j=1}^{N} \hat{p}_i^{j,x} \frac{1}{\lambda_p(C_j(\hat{\theta}) \cap K)} \int_{\mathbb{R}^{p-1}} \mathbb{1}_{C_j(\hat{\theta}) \cap K}(\theta) d\lambda_{p-1}(\theta) \\ =: \bar{p}_{\Theta_i}^{N,K}(\theta_i|X = x) d\lambda_1(\theta_i) \ , \end{aligned}$$

with the convention $\frac{1}{\lambda_p(C_j(\hat{\theta}) \cap K)} \mathbb{1}_{C_j(\hat{\theta}) \cap K} = 0$ if $\lambda_p(C_j(\hat{\theta}) \cap K) = 0$.

The Algorithm is summarized below :

Algorithm 5 Optimal Quantized Belief Propagation

Input : for $u = 1, \cdots, q$ and $x \in \mathcal{X}^n$, the factor $l_u(., x)$
Optimal Quantization : Prior's quantization
Belief Propagation : Computation of the quantized posterior using BP Algorithm with the quantized prior
Smoothing : Transformation of the quantized posterior into an absolutely continuous law
Output : A continuous marginal law $\pi_{\hat{\Theta}_i}(\cdot|X = x)$;

Getting the skills

Graphical Model On the group on which we focus, we apply the Optimal Quantized Belief Propagation on the TrueSkill factor with n children. We present in Figure 4.4 the graphical model used with three children for a better readability.

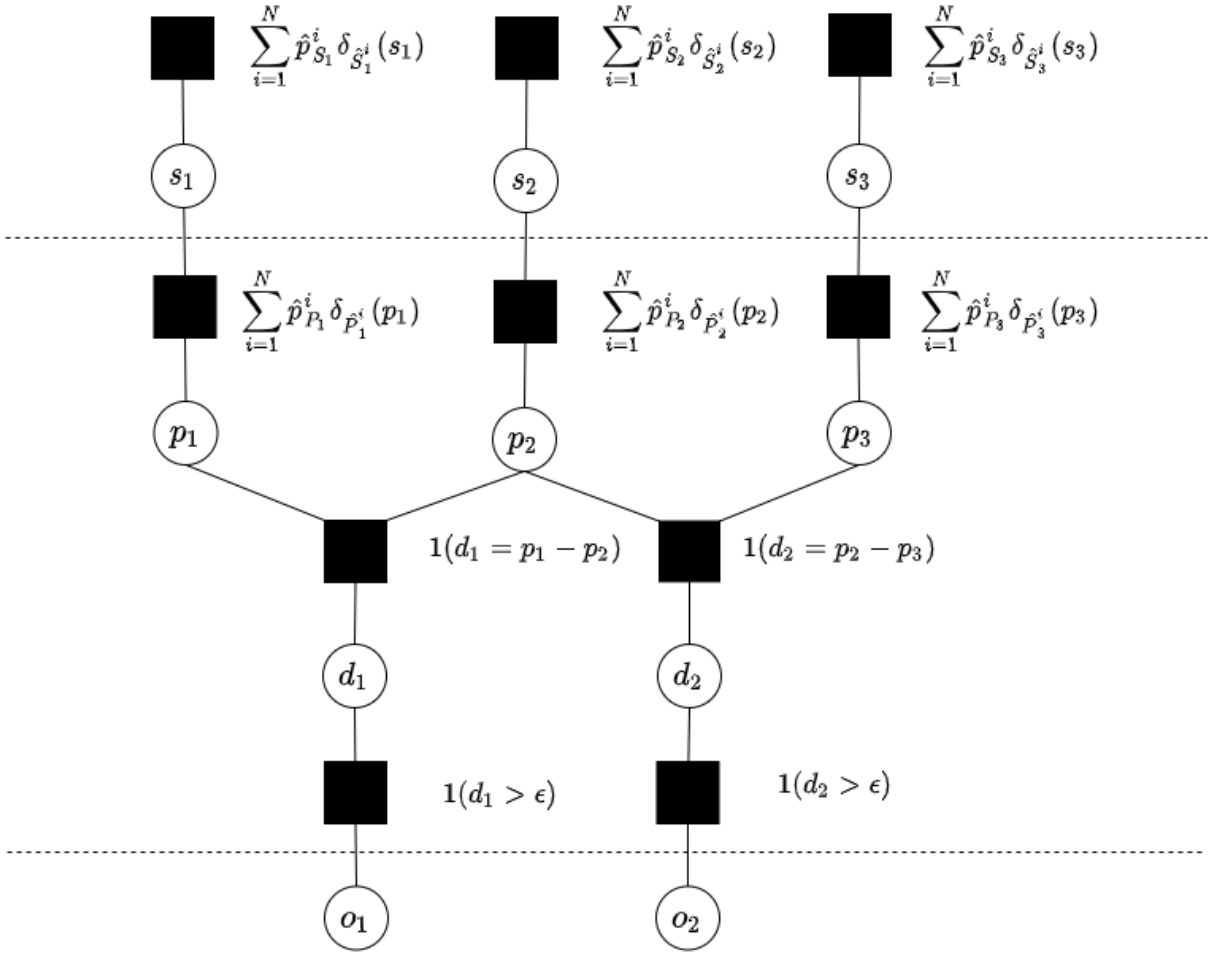
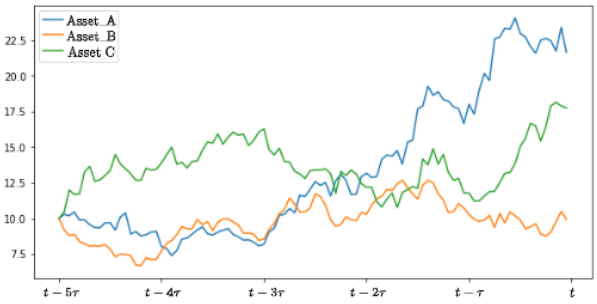


FIGURE 4.4 – Modified TrueSkill Graph with 3 players

Through observed ranking and Optimal Quantized Belief Propagation, each child's prior skill s_i, $\leq i \leq n$ is updated once.

Data preprocessing The preprocessing of data involves the following steps :

— At a time $t \geq 5\tau$ we compute from price data five non overlaping returns at time $t-4\tau, t-3\tau, t-2\tau, t-\tau$ and t as shown in Figure 4.5 with a group of three assets.



	Past Return Asset A	Past Return Asset B	Past Return Asset C
$t-4\tau$	-19.7%	-22.7%	38.9%
$t-3\tau$	2.1%	10.7%	17.3%
$t-2\tau$	60.9%	20.2%	-25.2%
$t-\tau$	36.7%	-0.01%	-3.3%
t	20.2%	-3.2%	50.6%

FIGURE 4.5 – Getting the past return

— From the past return, we compute five rankings as shown in Figure 4.6 with same instance as bellow.

	Past Return Asset A	Past Return Asset B	Past Return Asset C
$t-4\tau$	-19.7%	-22.7%	38.9%
$t-3\tau$	2.1%	10.7%	17.3%
$t-2\tau$	60.9%	20.2%	-25.2%
$t-\tau$	36.7%	-0.01%	-3.3%
t	20.2%	-3.2%	50.6%

	Asset A	Asset B	Asset C
$t-4\tau$	2	3	1
$t-3\tau$	3	2	1
$t-2\tau$	1	2	3
$t-\tau$	1	2	3
t	2	3	1

FIGURE 4.6 – Getting the past rankings

— From the five rankings, we apply the Optimal Quantized Belief Propagation algorithm on TrueSkill factor graph with the same initial gaussian prior $\mathcal{N}(25, 8^2)$ for each assets at time $t-4\tau$. This prior is quantized and updated five times sequentially using the five rankings derived below by applying the Optimal Quantized Belief Propagation Algorithm on TrueSkill graphical model. We present the modified TrueSkill graph at time t on Figure 4.7.

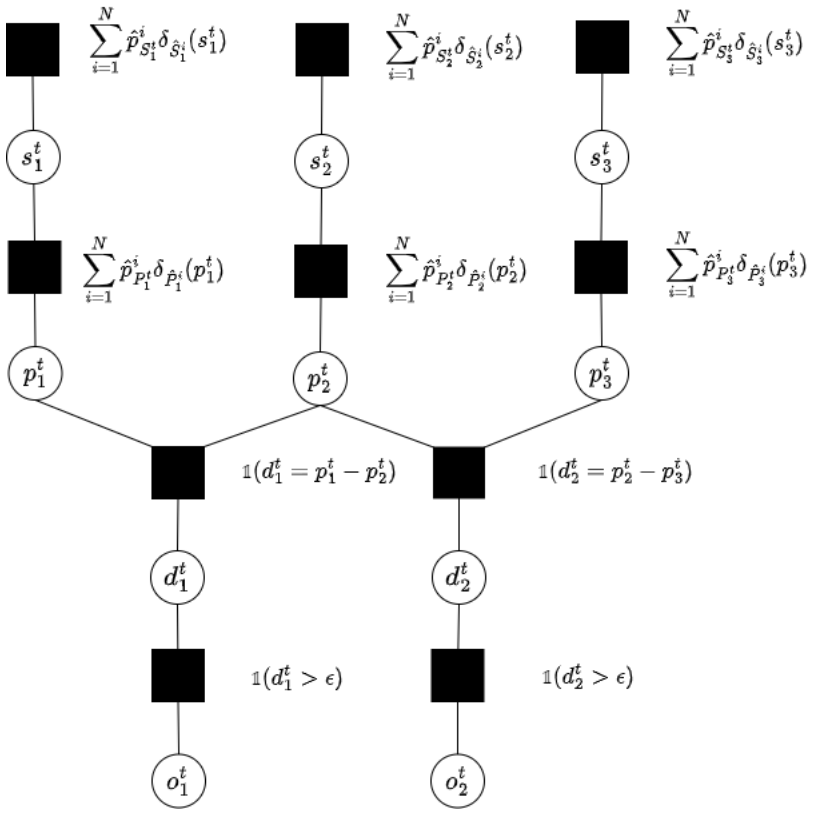


FIGURE 4.7 – Modified TrueSkill Graph

— From the last posterior law obtained at time t, we deduce the Skill Vector by setting $Z_i^t = \mathbb{E}[s_i^t],\ 1 \leq i \leq n$. An instance is presented in Figure 4.8 with $n = 3$.

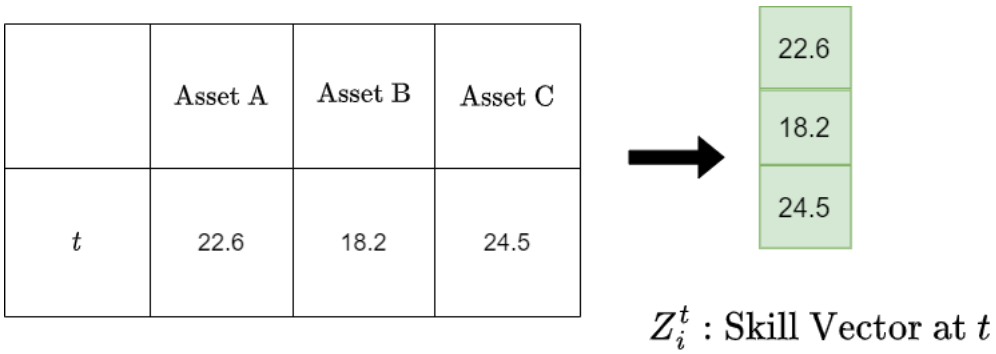


FIGURE 4.8 – Skill Vector Obtention

— This procedure is repeated for the entire history of the asset/children's group.

The whole history of each asset's skill is then obtained for every group of children.

4.4.3 The Prediction Model

Introducing the prediction model

The prediction model, represented in figure 4.9, takes as input the sequence of skills related to a group of children $\left(Z_i^1, \ldots, Z_i^{T_z}\right)$, defined in section 4.4.2, and predicts the rankings of the children over the next period of time.

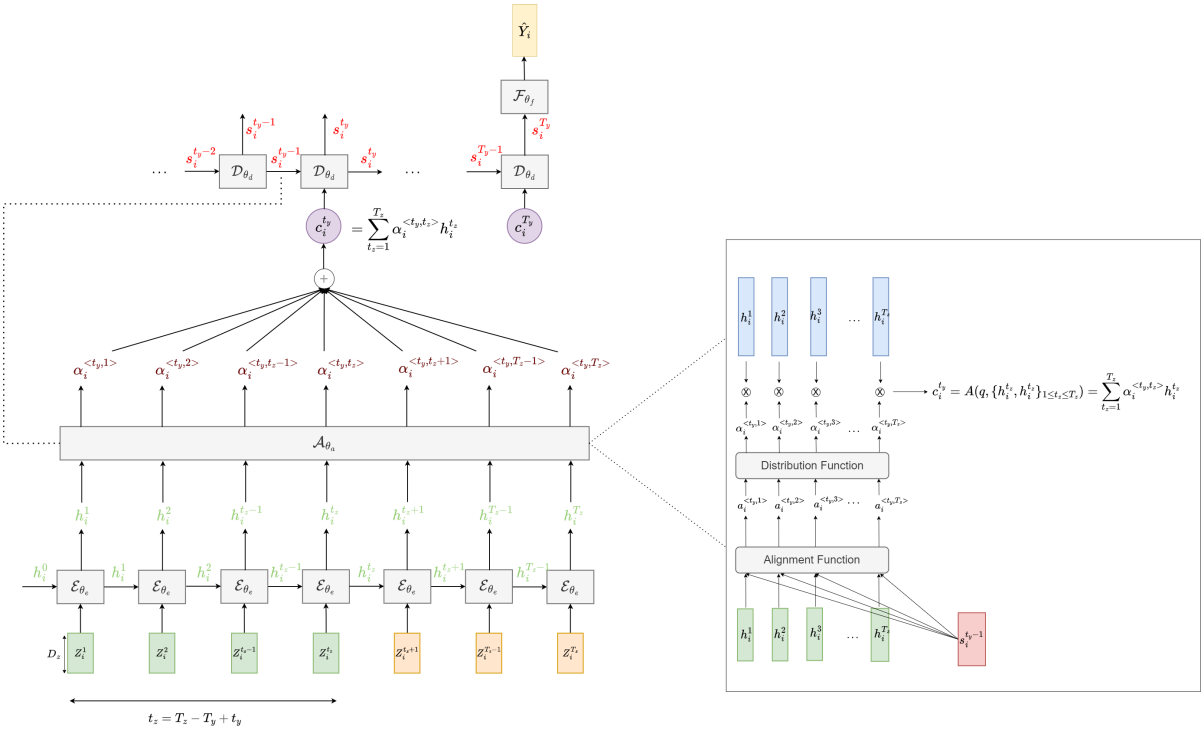


FIGURE 4.9 – The Prediction Model

The model is composed of the following layers :

— An Encoder layer.

— An Attention Layer.

— A Decoder layer.

— A Final Dense layer.

Let us now dive into each step of the model.

The encoder

In order to process the skills, we use a sequential model called the Gated Recurrent Units. The model, represented in 4.10 was introduced in 2014 (Cho et al., 2014) by Kyunghyun Cho.

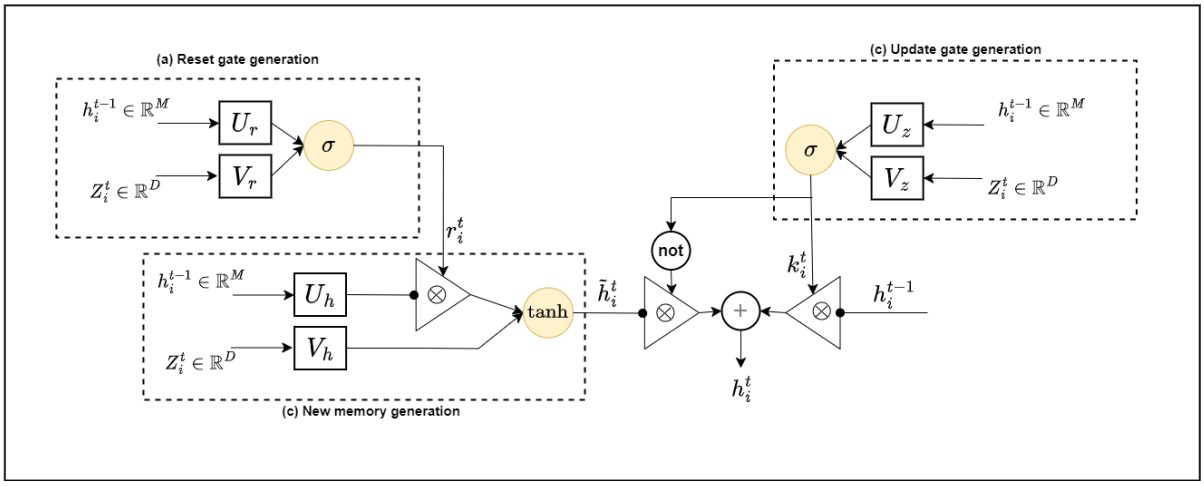


FIGURE 4.10 – The Gated Recurrent Units architecture

By processing the sequence of skill vectors one at a time, the GRU architecture aims to generate a sequence of hidden states $\left(h_i^1, \ldots, h_i^{T_z}\right)$ from a sequence of skills $\left(Z_i^1, \ldots, Z_i^{T_z}\right)$.

Consider $t \in \{2, \ldots, T_z\}$. The GRU generates the hidden state h_i^t based on the previous hidden state h_i^{t-1} and the new skill observation Z_i^t as follows :

— **Step (1) : Generating the reset gate vector**

The objective of the reset gate vector (eq 4.4.17) is to determine which dimensions of the previous hidden state h_i^{t-1} are relevant to the computation of the new memory candidate. It is computed by processing the previous hidden state h_i^{t-1} and the newly observed vector Z_i^t as follows :

$$r_i^t = \sigma\left(U_r h_i^{t-1} + V_r Z_i^t\right) \tag{4.4.17}$$

where σ stands for the sigmoid function

— **Step(2) : Generating the new memory candidate**

The new candidate $\tilde{h}_i^t$ is obtained by combining the information from the newly observed vector Z_i^t and the filtered version of the previous hidden state using the reset gate vector r_i^t. A point-wise $\tanh$ is then applied to push the values between -1 and 1 (eq 4.4.18).

$$\tilde{h}_i^t = \tanh\left(r_i^t \circ U_h h_i^{t-1} + V_h Z_i^t\right) \tag{4.4.18}$$

where $\circ$ stands for the Hadamard product.

— **Step (3) : Generating the update gate vector**

The update gate vector k_i^t is responsible for balancing between the information from the previous hidden state h_i^{t-1} and the information from the new memory candidate $\tilde{h}_i^t$. The vector is calculated using the previous hidden state h_i^{t-1} and the newly observed vector Z_i^t using the parameters U_k, V_k (eq 4.4.19)

$$k_i^t = \sigma\left(U_k h_i^{t-1} + V_k Z_i^t\right) \tag{4.4.19}$$

where σ stands for the sigmoid function

— **Step (4) : Getting the final hidden state**

The final hidden state h_i^t is generated using the update gate vector k_i^t. As showed in equation 4.4.20, if a particular dimension of k_i^t is close to 1, it means that almost all the information from the previous hidden state is copied out to h_i^t. Conversely, if it's close to 0, it means that the information from $\tilde{h}_i^t$ should be carried forward to h_i^t.

$$h_i^t = k_i^t \circ h_i^{t-1} + (1 - k_i^t) \circ \tilde{h}_i^t$$

The attention mechanisms

The Soft Query Retrieval Intuition Attention mechanisms originate from database Query-Retrieval Problems. Consider the database represented in figure 4.11 where a query is searched through the keys in order to retrieve a value (Garcia-Molina et al., 2000).

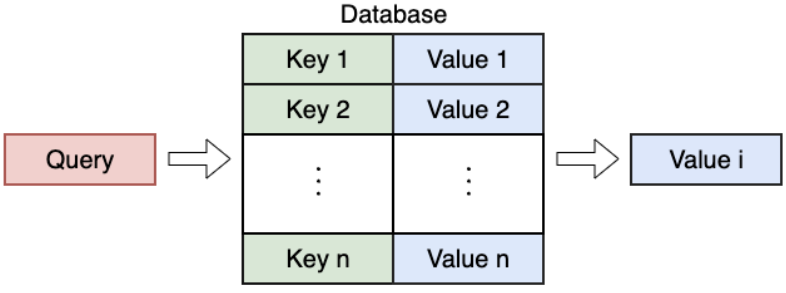


FIGURE 4.11 – Hard Query Retrieval Problem

Attention mechanisms can be viewed as a soft query retrieval process, where multiple keys can correspond to the query, rather than only one. As a result, we need to calculate the similarity between the query

and all the keys. The sum of the values, weighted by the computed similarities is the soft-query retrieval vector, or the **attention** vector.

Figure 4.12 represent the different steps involved in calculating the attention vector from a query $q \in \mathbb{R}^{d_q}$, a list of keys $(k_i)_{1\leq i\leq n} \in \mathbb{R}^{n\times d_k}$ and a list of values $(v_i)_{1\leq i\leq n} \in \mathbb{R}^{n\times d_v}$

— Using an alignment function a, we calculate the similarity between the query and all the keys as follows :

$$\forall i \in \{1,\dots,n\} \quad a_i = a(q,k_i)$$

— Several alignment functions have been proposed in the literature in order to get the alignment scores $(a_i)_{1\leq i\leq n}$, as shown in table 4.1 :

Function	**Equation**	**References**
Dot Product	$a(q,k_i) = q^T k_i$	(Luong et al., s. d.)
Scaled Dot Product	$a(q,k_i) = \frac{q^T k_i}{\sqrt{d_k}}$	(Vaswani et al., 2017)
Luong's Multiplicative alignment	$a(q,k_i) = q^T W k_i$	(Luong et al., s. d.)
Bahdanau's Additive alignment	$a(q,k_i) = v_a^T \tanh\left(W_1 q + W_2 k_i\right)$	(Bahdanau et al., 2017)
Feature-based	$a(q,k_i) = W_{imp}^T \text{act}(W_1\phi_1(k_i) + W_2\phi_2(q) + b)$	(Y. Li et al., 2019)
Kernel Method	$a(q,k_i) = \phi(q)^T\phi(k_i)$	(Yuan et al., 2021)

TABLE 4.1 – Alignment Functions

— The distribution function is used to map the alignment scores $(a_i)_{1\leq i\leq n}$ to the attention weights $(\alpha_i)_{1\leq i\leq n}$. The function ensures that the $\forall i \in \{1,\dots,n\} \quad \alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i = 1$.

— We usually use the softmax distribution function in order to get dense alignments as follows :

$$\forall i \in \{1,\dots,n\} \quad \alpha_i = \frac{e^{a_i}}{\sum_{j=1}^{n} e^{a_j}} \tag{4.4.20}$$

— Sparse alignment [1] can be obtained by using the **sparsemax** (Martins & Astudillo, 2016) or the **sparse entmax** (Martins et al., 2020) distribution functions.

— The attention vector $A\left(q,(k_i)_{1\leq i\leq n},(v_i)_{1\leq i\leq n}\right)$ is then calculated as follows :

$$A\left(q,(k_i)_{1\leq i\leq n},(v_i)_{1\leq i\leq n}\right) = \sum_{i=1}^{n} \alpha_i v_i$$

1. Nonzero probabilities are assigned to only a few number of values

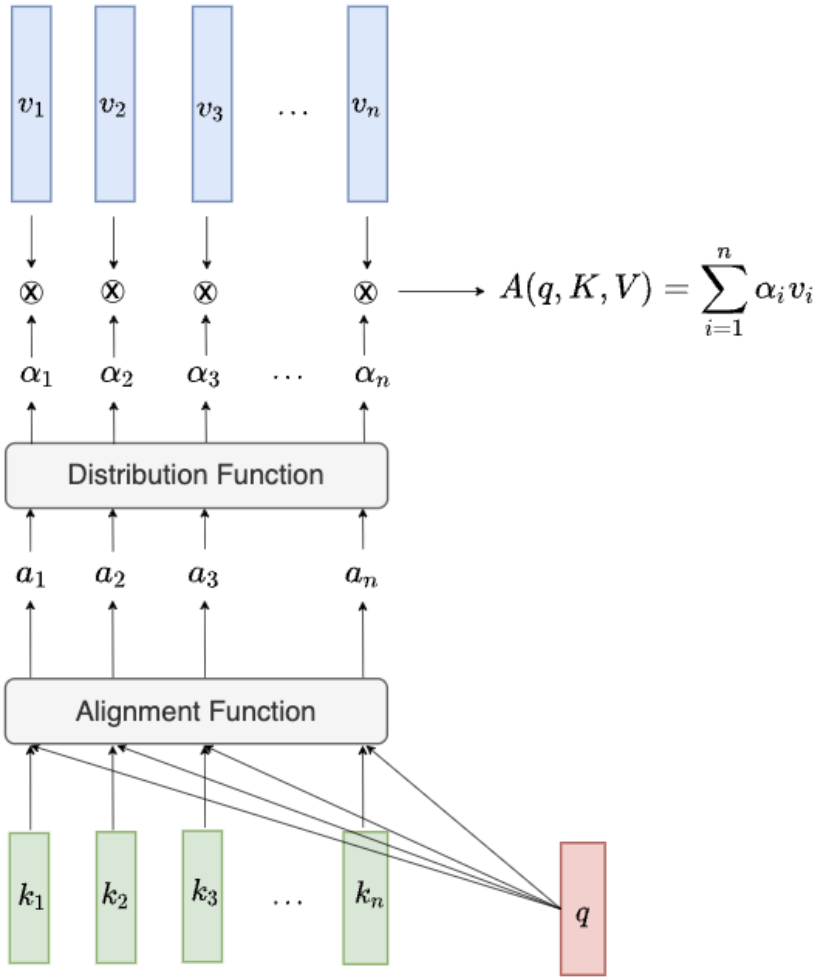


FIGURE 4.12 – Soft-Query Retrieval

The attention layer Consider a decoder time step $t_y \in \{1, \dots, T_y\}$. The attention layer $\mathcal{A}_{\theta_a}$, represented in figure 4.13, assigns a weight to each encoder hidden state $(h_i^{t_z})_{1 \leq t_z \leq T_z}$ in order to output the final **context vector** $c_i^{t_y}$ as follows :

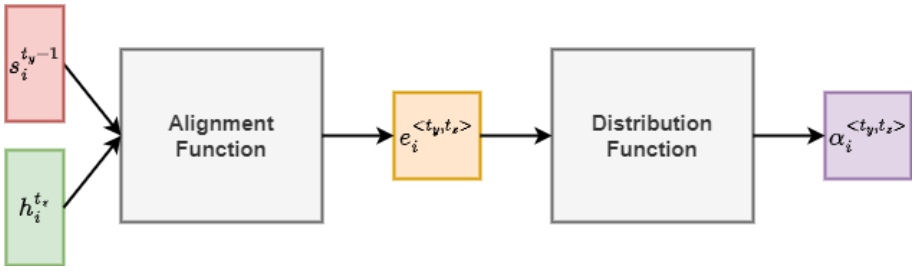


FIGURE 4.13 – The attention layer

— A Feed Forward neural network is applied to calculate an alignment score between the decoder previous hidden state $s_i^{t_y-1}$ and each encoder hidden state $h_i^{t_z} \in \{h_i^1, \dots, h_i^{T_z}\}$. There are multiple alignment functions available, including the additive alignment function of Bahdanau (Bahdanau

et al., 2017), as well as the multiplicative alignment function of Luong (Luong et al., s. d.).

$$e_i^{<t_y,t_z>} = \begin{cases} v_a^T \tanh\left(W_1 s_i^{t_y-1} + W_2 h_i^{t_z}\right) & \text{Bahdanau's additive alignment function} \\ {s_i^{t_y-1}}^T W h_i^{t_z} & \text{Luong's multiplicative alignment function} \end{cases}$$

— The resulting alignment score $e_i^{<t_y,t_z>}$ can then be turned into the attention weight $\alpha_i^{<t_y,t_z>}$ using a distribution function (such as the softmax, the sparsemax (Martins & Astudillo, 2016) or the sparse entmax (Martins et al., 2020) distribution functions).

— The attention vector (also called the context vector) is then calculated as follows :

$$c_i^{t_y} = \sum_{t_z=1}^{T_z} \alpha_i^{<t_y,t_z>} h_i^{t_z}$$

— As a result, the context vector can be seen as the attention vector associated with the query $q = s_i^{t_y-1}$, the keys $(k_t)_{1\leq t\leq T_z} = (h_i^{t_z})_{1\leq t_z\leq T_z}$ and the values $(v_t)_{1\leq t\leq T_z} = (h_i^{t_z})_{1\leq t_z\leq T_z}$.

$$A\left(q, (k_i)_{1\leq i\leq n}, (v_i)_{1\leq i\leq n}\right) = \sum_{t_z=1}^{T_z} \alpha_i^{<t_y,t_z>} h_i^{t_z} = c_i^{t_y}$$

Masking future skills At each time step $t_y \in \{1, \ldots, T_y\}$, we would like the context vector to depend only on the first skills $\{1, \ldots, t_z\}$ where $t_z = T_z - T_y + t_y$ in order to avoid any look ahead bias, as shown in the figure 4.14

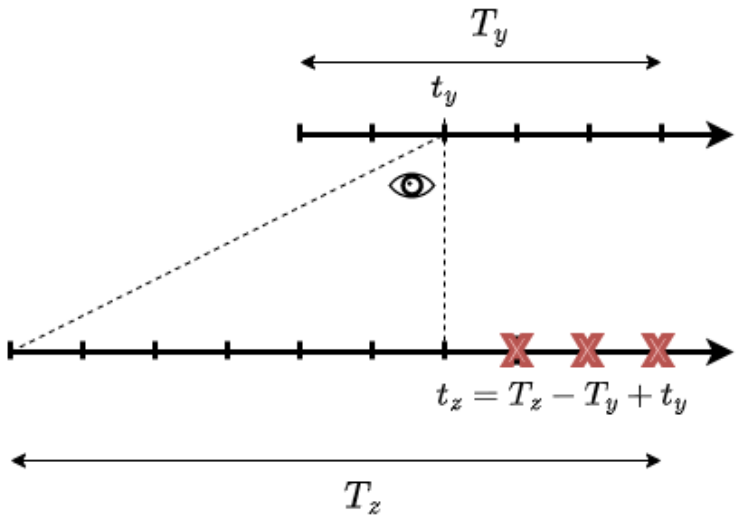


FIGURE 4.14 – Applying a mask

To that end, for all $t_y \in \{1, \ldots, T_y\}$, we set $e^{<t_y,t>}$, to $-\infty$ for all $t > t_z$, as shown in the following

table :

	1	2	…	t_z	…	$T_z - 1$	T_z
1	$e_i^{<1,1>}$	-∞	…	-∞	…	-∞	-∞
2	$e_i^{<2,1>}$	$e_i^{<2,2>}$	…	-∞	…	-∞	-∞
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
t_y	$e_i^{<t_y,1>}$	$e_i^{<t_y,2>}$	…	$e_i^{<t_y,t_z>}$	…	-∞	-∞
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$T_y - 1$	$e_i^{<T_y-1,1>}$	$e_i^{<T_y-1,2>}$	…	$e_i^{<T_y-1,t_z>}$	…	$e_i^{<T_y-1,T_z-1>}$	-∞
T_y	$e_i^{<T_y,1>}$	$e^{<T_y,2>}$	…	$e_i^{<T_y,t_z>}$	…	$e_i^{<T_y,T_z-1>}$	$e_i^{<T_y,T_z>}$

After applying the softmax function, we get the masked attention weights :

	1	2	…	t_z	…	$T_z - 1$	T_z
1	$\alpha_i^{<1,1>}$	0	…	0	…	0	0
2	$\alpha_i^{<2,1>}$	$\alpha_i^{<2,2>}$	…	0	…	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
t_y	$\alpha_i^{<t_y,1>}$	$\alpha_i^{<t_y,2>}$	…	$\alpha_i^{<t_y,t_z>}$	…	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$T_y - 1$	$\alpha_i^{<T_y-1,1>}$	$\alpha_i^{<T_y-1,2>}$	…	$\alpha_i^{<T_y-1,t_z>}$	…	$\alpha_i^{<T_y-1,T_z-1>}$	0
T_y	$\alpha_i^{<T_y,1>}$	$\alpha^{<T_y,2>}$	…	$\alpha_i^{<T_y,t_z>}$	…	$\alpha_i^{<T_y,T_z-1>}$	$\alpha_i^{<T_y,T_z>}$

The decoder

The decoder is also a GRU model, parameterized by the following parameters : $\tilde{U}_r \in \mathbb{R}^{M\times M}, \tilde{V}_r \in \mathbb{R}^{D\times M}, \tilde{U}_s \in \mathbb{R}^{M\times M}, \tilde{V}_s \in \mathbb{R}^{D\times M}, \tilde{U}_k \in \mathbb{R}^{M\times M}, \tilde{V}_k \in \mathbb{R}^{D\times M}$. The hidden states are denoted $(s_i^{t_y})_{1\leq t_y\leq T_y}$.

At each time step $t_y \in \{2, \ldots, T_y\}$, the decoder $\mathcal{D}_{\theta_d}$ outputs the hidden state $s_i^{t_y}$ based on the previous hidden state $s_i^{t_y-1}$ and the context vector $c_i^{t_y}$, as follows :

$$r_i^{t_y} = \sigma\left(\tilde{U}_r s_i^{t_y-1} + \tilde{V}_r c_i^{t_y}\right) \quad \text{(Generating the reset gate vector)}$$
$$\tilde{s}_i^{t_y} = \tanh\left(r_i^{t_y} \circ \tilde{U}_s s_i^{t_y-1} + \tilde{V}_s c_i^{t_y}\right) \quad \text{(Generating the new memory candidate)}$$
$$k_i^{t_y} = \sigma\left(U_k s_i^{t_y-1} + V_k c_i^{t_y}\right) \quad \text{(Generating the update gate vector)}$$
$$s_i^{t_y} = (1 - k_i^{t_y}) \circ s_i^{t_y} + k_i^{t_y} \circ s_i^{t_y-1} \quad \text{(Generating the new hidden state)}$$

The Final Layer :

The final layer $\mathcal{F}_{\theta_f}$ is then applied on the final hidden state $s_i^{T_y}$ in order to generate the prediction $\hat{Y}_i$ as follows :

$$\hat{Y}_i = \text{softmax}\left(W_f^T s_i^{T_y} + b_f\right)$$

4.4.4 The Optimization Process

Creating the training dataset

Consider D child assets associated with a parent node in the tree structure 4.2.

Consider a training dataset $\mathcal{T} = \{(Y_i, \hat{Y}_i)_{1 \leq i \leq N}$, where $\hat{Y}_i$ is the output of the prediction model associated with the sequence of skills $\left(Z_i^1, \ldots, Z_i^{T_z}\right)$ and Y_i represents the vector of returns over the next period of time for all the assets associated with the parent node.

Let $\theta = (\theta_e, \theta_a, \theta_d, \theta_f)$ represent the whole set of parameters related to all the layers.

Introducing the optimization problem

The objective of the ranking system is to minimise a loss function $\mathcal{L}$ associated with the Ranking problem.

$$\min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\hat{Y}_{(i)}, Y_{(i)})$$

We have considered two loss functions : The ListNet loss function and the RankCosine loss function.

The optimization problem, can be written as follows :

$$\min_{\theta \in \mathbb{R}^d} f(\theta) := \mathbb{E}_{s \sim \mathbb{P}}[\mathcal{L}(\theta, s)], \tag{4.4.21}$$

where :

— The function f is called **the objective function**.

— The function $\mathcal{L}$ is the loss function.

— $\mathbb{P}$ is the unknown data distribution on the domain $\mathcal{S}$

— θ is the set of parameters we wish to optimize.

In the following sections, we are going to focus on the Adam optimizer. Section 4.4.4 introduces a description of the algorithm. In section 2.7.4, we analyze the convergence behavior of the algorithm in the nonconvex setting.

Description of the algorithm

The Adam algorithm was first introduced in 2015 (Kingma & Ba, 2015). The authors proposed a proof of convergence which was found to have problems. In 2018, (S. J. Reddi et al., 2018) clarified the inconsistency of the previous paper and fixed the proof in the convex setting. (S. Reddi et al., 2018) conducted the proof for the non convex case under some useful parameter settings.

The Adam algorithm defined in (S. Reddi et al., 2018) is summarized in Algorithm 6

Algorithm 6 The Adam Optimizer

Input:
Initial parameter value : $\theta_1 \in \mathbb{R}^d$
Learning rates : $\{\eta_t\}_{t=1}^T$
Decay parameters : $0 \leq \beta_1, \beta_2 \leq 1$
Stability parameter : $\epsilon > 0$
Output: ϵ-First Order Stationary Point θ_{T+1}

Set $m_0 = 0, v_0 = 0$
for $t = 1$ **to** T **do**
  Draw a batch $(s_t^i)_{i \in \mathcal{B}_t}$ from $\mathbb{P}$
  Compute $g_t = \frac{1}{|\mathcal{B}_t|} \sum_{s \in \mathcal{B}_t} \nabla \mathcal{L}(\theta_t, s)$
  Update $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t$
  Update $v_t = v_{t-1} - (1 - \beta_2)\,(v_{t-1} - g_t \circ g_t)$
  Update $\theta_{t+1} = \theta_t - \eta_t \frac{m_t}{\sqrt{v_t + \epsilon}}$

4.5 Empirical Results

4.5.1 Signature

To define the trading strategy, we need to introduce the concept of signature. Each stocks is a component of a industry which is also a component of a sector. So we can divide a whole stock index into the following tree structure :

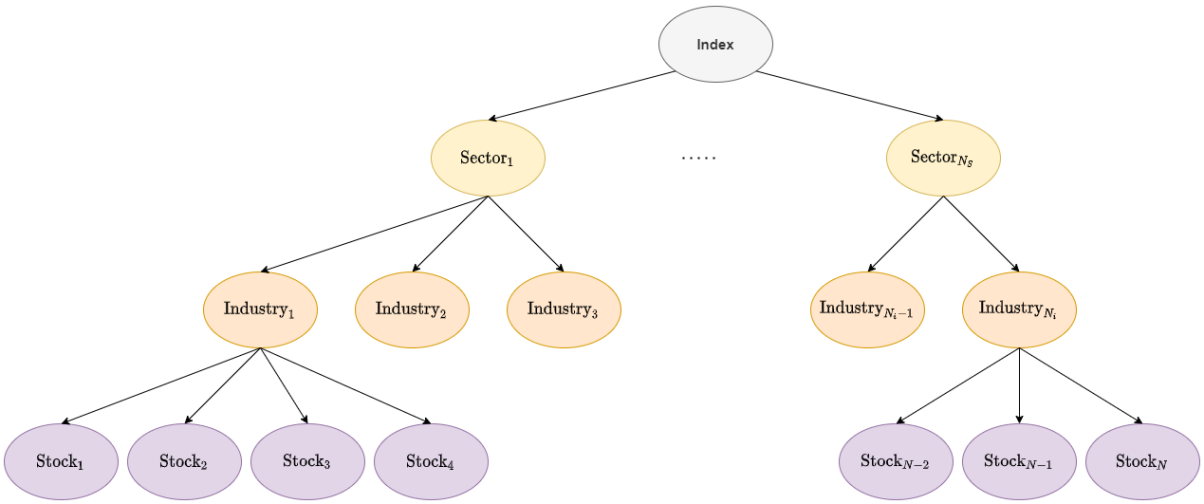
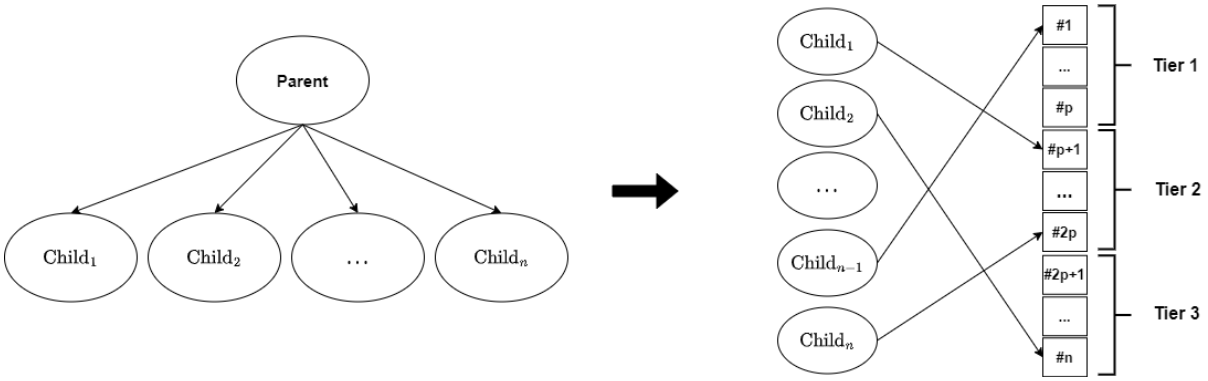


FIGURE 4.15 – Index Tree

So each day, we have access to the predicted ranking of the sector among its peer groups of sector, the predicted ranking of the industry among its peers and stocks among its peers. We can divide the predicted ranking into three tiers [2] as explained into the following picture.



Click to show the PDF

FIGURE 4.16 – Tiers Prediction

We can define hence a stock signature by the triplet composed by the tiers of sectors, tiers of industry and tiers of stock.

2. If the len of the groups does not divide evenly by three, and the remainder of the Euclidean division is equal to one, the remainder child is placed in the second tier, and if the remainder of the Euclidean division is equal to two, the two remainders children are placed in the first and third tiers.

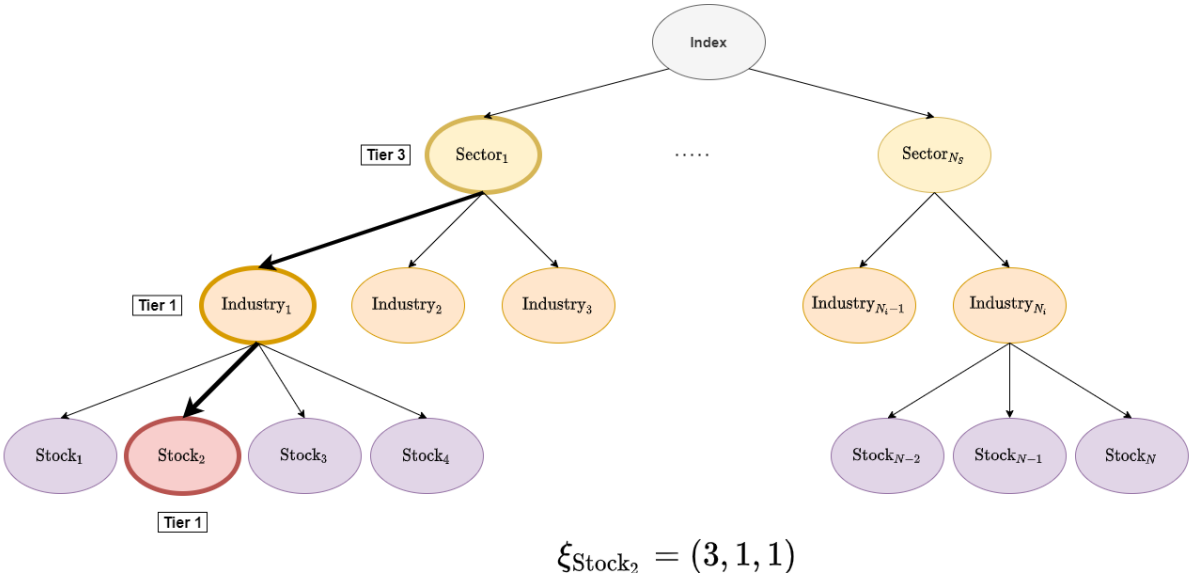


FIGURE 4.17 – Stock's Signature

4.5.2 Data processing

We begin testing by 2015 and we want at least for each groups 5 years of historical data. At worst, the first Training period for each group begins in January 2010 and ends in December 2014, and the first Testing period is in 2015. We use an expanding window approach for each year until 2021. We illustrate this in Figure 4.18 by fixing the first period to T_0.

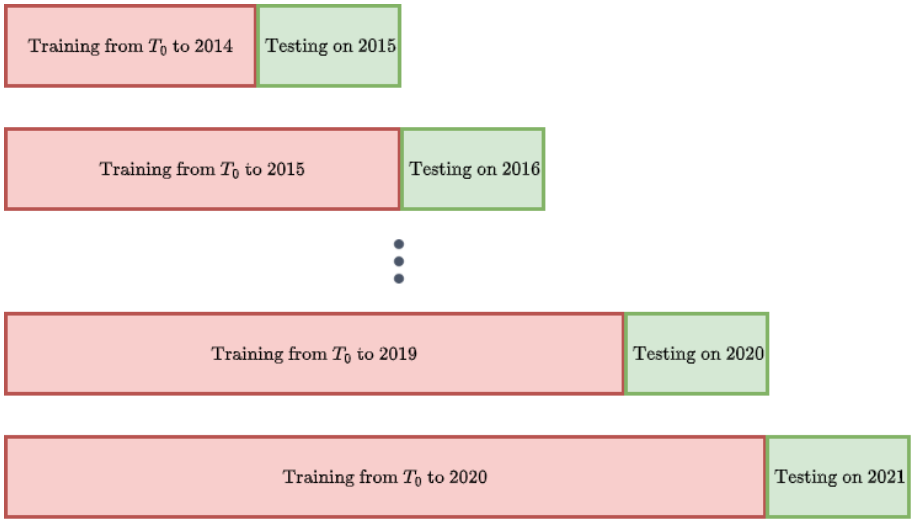


FIGURE 4.18 – Training - Testing Procedure

Details of asset data can be found in Appendix B.4

4.5.3 Benchmark

We chose to benchmark the proposed ranking trading strategy against a cross-sectional momentum strategy inspired by (Jegadeesh & Titman, 1993), a momentum ranking strategy described in (Clenow, 2015) and an uniform allocation.

Cross Sectional Momentum

Each time t, we rank the assets based on their return from $t-T$ to $t-1$. In particular, if we denote by S_t the price at time t of an asset then we compute the following return at each time t, the following return

$$\hat{s}_t = \frac{S_{t-1} - S_{t-T}}{S_{t-T}}$$

. Rankings are derived from these scores for each group. As done in literature, we consider two versions, $T = 126$ days (six trading months) and $T = 252$ (12 trading months).

Ranking Momentum

Each time t, we rank the assets by computing the exponential regression of assets that we obtain from a linear regression of log return on daily close of the asset. Mathematically, we compute each time t, the following score momentum :

$$\hat{s}_t = (1 + \hat{\alpha}_t)^{252} R_t^2$$

where $\hat{\alpha}_t$ is the slope obtained with mean squared minimization :

$$(\hat{\alpha}_t, \hat{\beta}_t) = \arg\min_{\alpha,\beta} \sum_{i=t-T}^{t} \left(\log(\frac{P_i}{P_{i-1}}) - (\alpha i + \beta) \right)^2$$

and

$$R_t = 1 - \frac{\sum\limits_{i=t-T}^{t} \big(X_i - (\hat{\alpha} i + \beta)\big)^2}{\sum\limits_{i=t-T}^{t} \big(X_i - \frac{1}{T} \sum\limits_{j=t-T}^{t} (\hat{\alpha} j + \hat{\beta})\big)^2}$$

We fix $T = 100$ days.

Uniform allocation benchmark

Each time t, we average all stocks with a complete signature.

4.5.4 Empirical Results

Across all benchmarks, we calculate the score iteratively at the sector, industry, and stock levels, and we form three clusters : A, B, and C. Cluster A corresponds to stocks with a sum of signatures less than or equal to 4. The cluster B contains stocks with a sum of signatures between 5 and 7. The cluster C contains the others stocks, i.e stocks with a sum of signatures 8 and 9. We consider equal weighted allocation for each strategy. To be clear, we present at Figure 4.19 the definition of Cluster A, B, C.

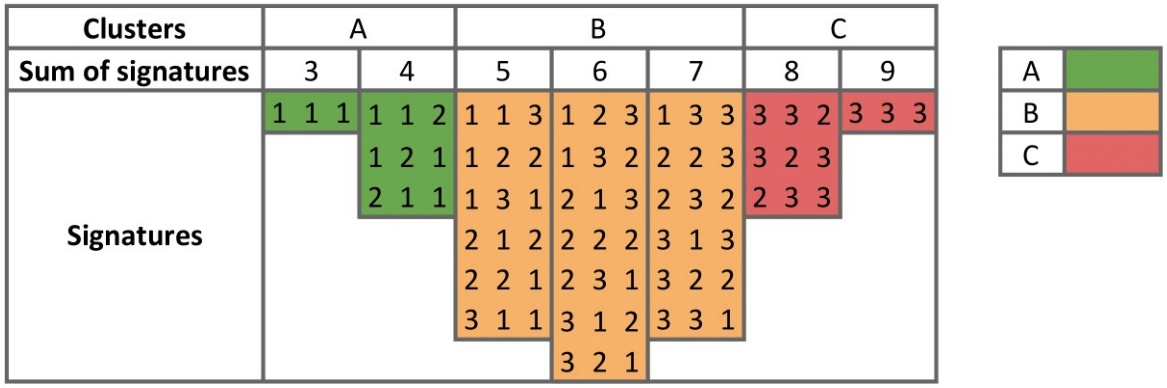


FIGURE 4.19 – Clusters A, B, C

Evaluation of performance of the various trading algorithms is done by computing the following financial metrics :

- The Annualised Return R
- The Annualised Volatility σ
- The Sharpe ratio ($\frac{R}{\sigma}$)
- The Drawdown MDD which is the worst peak-to-trough decline
- The Sortino Ratio which is the ratio between Annualised Return and Standard Deviation of negative Returns
- The Calmar Ratio ($\frac{R}{MDD}$)
- The percentage of daily positive returns

We summarize our results for the Cluster A on Table 4.2 and we refer to Appendix B.2 for detailed results.

	Benchmark				Proposed Model	
	Cross Sectional Momentum (T=126)	Cross Sectional Momentum (T=252)	Ranking Momentum	Uniform	RankCosine	ListNet
Annualized Return	15,91	15,49	16,25	14,61	**18,96**	18,73
Annualized Volatility	**18,91**	19,11	20,29	19,8	20,78	20,66
Sharpe Ratio	0,84	0,81	0,80	0,74	**0,91**	**0,91**
MDD	-34,50	**-33,22**	-40,28	-39,76	-38,04	-38,04
Sortino Ratio	1,02	0,98	0,94	0,87	**1,08**	**1,08**
Calmar Ratio	0,46	0,47	0,40	0,37	**0,50**	**0,50**
% Daily Positive Return	56,03	**56,20**	55,63	54,72	55,17	55,63

TABLE 4.2 – Results for Cluster A

In comparison with benchmark models, the proposed models have much improved annualized returns, Sharpe ratios, Sortino ratios, and Calmar ratios.

We can also decompose deeply the cluster into $A+$ and A-, C+ and C-, as shown on the following Figure 4.20.

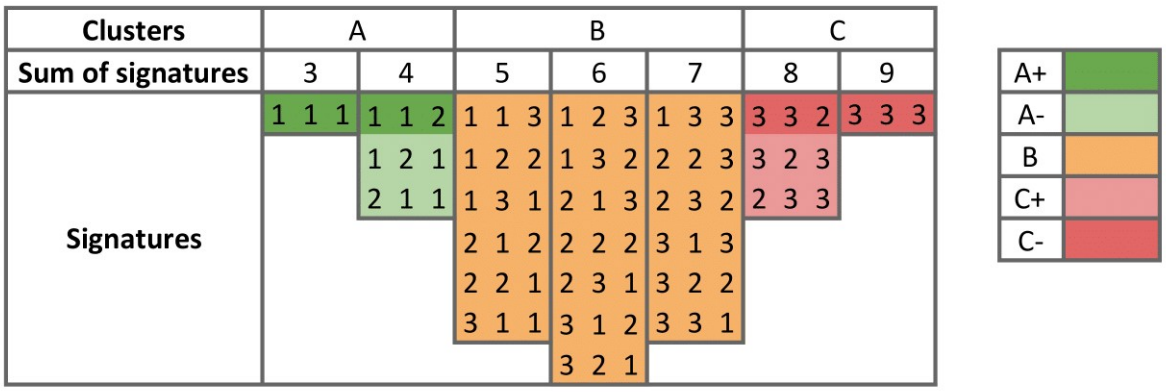


FIGURE 4.20 – Clusters $A+$, $A-$, B, C+, C-

The Table 4.3 presents the results by go deeply into cluster by focusing on Cluster A+ as defined below. The detailed comparison are presented in Appendix.

	Benchmark				Proposed Model	
	Cross Momentum (T=126)	Cross Momentum (T=252)	Ranking Momentum	Uniform	RankCosine	ListNet
Annualized Return	13,28	12,14	17,18	14,61	**21,23**	20,45
Annualized Volatility	20,19	20,19	21,29	**19,8**	21,28	21,57
Sharpe Ratio	0,66	0,60	0,81	0,74	**0,99**	0,95
MDD	-32,93	**-32,91**	-39,12	-39,76	-39,43	-39,36
Sortino Ratio	0,82	0,73	0,99	0,87	**1,18**	1,14
Calmar Ratio	0,40	0,37	0,44	0,37	**0,54**	0,52
% Daily Positive Return	54,49	55,69	56,20	54,72	**57,18**	**57,18**

TABLE 4.3 – Results for Cluster A+

	Benchmark				Proposed Model	
	Cross Momentum (T=126)	Cross Momentum (T=252)	Ranking Momentum	Uniform	RankCosine	ListNet
Annualized Return	13,55	10,06	**6,21**	14,61	9,01	9,73
Annualized Volatility	28,32	**29,16**	23,06	19,8	23,44	23,43
Sharpe Ratio	0,48	0,35	**0,27**	0,74	0,38	0,42
MDD	-49,37	-50,49	**-53,16**	-39,76	-44,59	-44,59
Sortino Ratio	0,63	0,46	**0,32**	0,84	0,50	0,53
Calmar Ratio	0,27	0,20	**0,12**	0,37	0,20	0,22
% Daily Positive Return	**51,4**	51,52	52,66	54,72	51,97	51,74

TABLE 4.4 – Results for Cluster C-

The results of our proposed model are leveraged with a Cluster A+ and Cluster C- that reflects expected behavior, which is Cluster A+ with good results and Cluster C- with poor results. In terms of Returns, Sharpe ratio, Sortino ratio and Calmar ratio, Cluster A+ of our proposed model is the best. More generally, the proposed model presents a better differentiation of clusters based on return and Sharpe ratio. Other benchmarks do not verify this. It is possible to improve sharpe ratios and drawdowns if volatility is controlled, as shown in Appendix B.3.

4.6 Conclusion

By learning to focus on relevant parts of the skill vectors, the Ranking System is able to forecast the different tiers over the next period of time. As a result, we can define the signature associated with each of the 500 largest US stocks, enabling us to develop very intuitive strategies. The proposed strategy, which clusters the universe based on their signatures, outperforms both the market and traditional momentum

strategies substantially.

A | Forecasting Market turbulence regimes using Latent Spectral representation

A.1 The Time-Frequency Localization information

We need to introduce the concept of a **window function** to achieve the time frequency localization of a time varying signal.

A window function is a function $\phi(.)$ in $\in \mathcal{L}^2(\mathbb{R})$ such that both the function $t \mapsto \phi(t)$ and its Fourier transform $\omega \mapsto \hat{\phi}(\omega)$ have rapid decay. Therefore, multiplying a signal by a window function enables us to correlate the signal with a waveform well localized in time and in frequency, and thus retrieve the spectral information of the signal in the domain of influence of the window function. By translating the window function on the time axis, we can cover the entire time domain. The waveforms are called *time-frequency atoms*.

Let us consider a general dictionary of atoms $\mathcal{D} = \{\phi_\gamma\}_{\gamma\in\Gamma}$ where $\phi_\gamma \in \mathcal{L}^2(\mathbb{R})$ and γ represents a multiindex parameter. We define the linear time frequency transform of a signal $f \in \mathcal{L}^2(\mathbb{R})$ as follows :

$$\mathcal{T}f(\gamma) = \langle f, \phi_\gamma \rangle = \int_{-\infty}^{+\infty} f(t)\phi_\gamma^*(t)dt \tag{A.1.1}$$

Example : If $\gamma = (m, \xi)$ represents the time-frequency plane, the idea in the equation A.1.1 is to make $\phi_\gamma(.)$ nearly zero outside a neighborhood of m and $\hat{\phi}_\gamma(.)$ nearly zero outside a neighborhood of ξ. Then, $\langle f, \phi_\gamma \rangle$ reveals the properties of the signal f in the neighborhood of m and the properties of $\hat{f}$ in the neighborhood of ξ.

In the following section, we will try to quatify the slice of information in the time frequency plane provided by $\langle f, \phi_\gamma \rangle$. For this, we introduce the Heisenberg boxes.

A.1.1 Heisenberg Boxes

We choose ϕ_γ such that $||\phi_\gamma||_2^2 = 1$. We can then interpret $|\phi_\gamma(.)|^2$ as a probability distribution. Let us consider its temporal mean m_γ and its temporal variance $\sigma_t^2(\gamma)$ as follows :

$$m_\gamma = \int_{-\infty}^{+\infty} t|\phi_\gamma(t)|^2 dt \quad \text{and} \quad \sigma_t^2(\gamma) = \int_{-\infty}^{+\infty} (x - m_\gamma)^2 |\phi_\gamma(x)|^2 dx \tag{A.1.2}$$

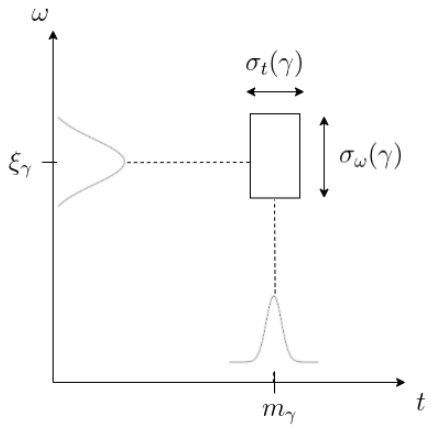


FIGURE A.1 – Heisenberg Box

By using the following *Plancherel* formula :

$$\forall u \in \mathcal{L}^1(\mathbb{R}) \cap \mathcal{L}^2(\mathbb{R}) \quad \int_{-\infty}^{+\infty} |u(t)|^2 dt = \frac{1}{2\pi} \int_{-\infty}^{+\infty} |\hat{u}(\omega)|^2 d\omega \tag{A.1.3}$$

We conclude that :

$$\int_{-\infty}^{+\infty} \frac{1}{2\pi} |\hat{\phi_\gamma}(\omega)|^2 d\omega = ||\phi_\gamma||_2^2 = 1 \tag{A.1.4}$$

Therefore, $\frac{1}{2\pi}|\hat{\phi}_\gamma(.)|^2$ can also be seen as a probability distribution centered around the frequency mean ξ_γ with a variance $\sigma_\omega^2(\gamma)$ defined as follows :

$$\xi_\gamma = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \omega |\hat{\phi}_\gamma(\omega)|^2 d\omega \quad \text{and} \quad \sigma_\omega^2(\gamma) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} (\omega - \xi_\gamma)^2 |\hat{\phi}_\gamma(\omega)|^2 d\omega \tag{A.1.5}$$

The time-frequency resolution of ϕ_y can then be represented in the time-frequency plane (t, ω) by a Heisenberg box centered at (m_γ, ξ_γ) with a spread around m_γ equal to $\sigma_t(\gamma)$ and a spread around ξ_γ equal to $\sigma_\omega(\gamma)$.

The figure A.1 shows the Heisenberg box characterizing the time-frequency resolution of ϕ_γ.

The Heisenberg uncertainty theorem [1] shows that there is a compromise to find between the time resoltuion and the freqency resolution :

1. which proves that $\sigma_t(\gamma)\sigma_\omega(\gamma) \geq \frac{1}{2}$

We will focus now on a particular time frequency atom introduced by Gabor in 1946. It's called *the windowed Fourier atom*, which is constructed with a window function g modulated by the frequency ξ and translated by m.

A.1.2 The Gabor Transform

We have defined the Heisenberg boxes as a way to quantify the time-frequency localization of general atoms ϕ_γ, we now need to extract useful information from the signal by correlating it with a specific type of atoms, well-localized in the time-frequency space, called *the windowed Fourier atoms*.

Let g be a real and symmetrical [2] window function (such that $||g||_2^2 = 1$). The windowed Fourier atoms are defined as follows :

$$\forall t \in \mathbb{R} \quad \phi_\gamma(t) = g_{m,\xi}(t) = e^{i\xi t} g(t-m)$$

Let g be a real and symmetrical [3] window function (such that $||g||_2^2 = 1$). The windowed Fourier atoms are defined as follows :

$$\forall t \in \mathbb{R} \quad \phi_\gamma(t) = g_{m,\xi}(t) = e^{i\xi t} g(t-m)$$

The resulting atom dictionary $\mathcal{D} = \{\phi_\gamma\}_{\gamma\in\Gamma} = \{g_{m,\xi}\}_{(m,\xi)\in\mathbb{R}^2}$ is both translation-invariant [4] and frequency invariant [5]. Therefore, it is particularly useful to analyse patterns which are translated in time and frequency.

Furthermore, $\forall (m,\xi) \in \mathbb{R}^2 \quad ||g_{m,\xi}||_2^2 = 1$. The resulting time frequency transform (called the Gabor transform) of a signal $f \in \mathcal{L}^2(\mathbb{R})$ is defined as follows :

$$\mathcal{G}f(m,\xi) = \langle f, g_{m,\xi}\rangle = \int_{-\infty}^{+\infty} f(t) e^{-i\xi t} g(t-m) dt \tag{A.1.6}$$

In order to measure the energy of the signal in a time-frequency neighborhood of (m,ξ), we define the *spectrogram* denoted $\mathcal{S}f$ as follows :

2. i.e, $\forall t \in \mathbb{R} \quad g(-t) = g(t)$
3. i.e, $\forall t \in \mathbb{R} \quad g(-t) = g(t)$
4. A translation-invariant representation is obtained when for all m the function $t \mapsto \phi_\gamma(t+m)$ is still in $\mathcal{D}$.
5. Since : $\forall \xi' \in \mathbb{R} \quad \hat{g}_{m,\xi}(\omega - \xi') = \mathcal{F}\left(t \mapsto e^{i\xi' t} g_{m,\xi}(t)\right) = \mathcal{F}\left(t \mapsto e^{i(\xi'+\xi)t} g(t-m)\right) = \hat{g}_{m,\xi'+\xi}(\omega)$

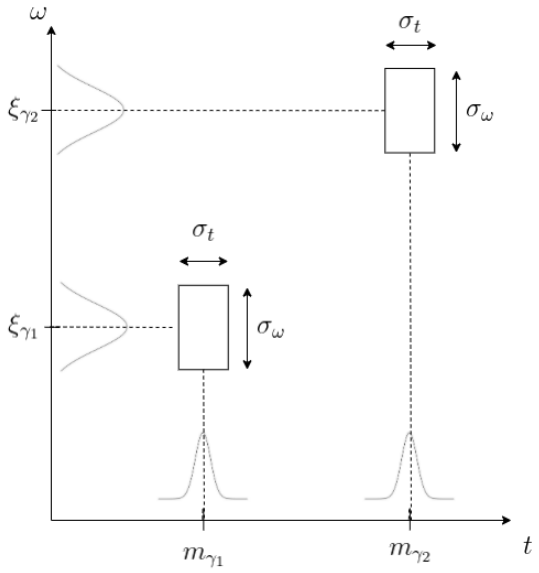


FIGURE A.2 – Heisenberg Box for the windowed Fourier atoms

$$\mathcal{S}f(m,\xi) = |\mathcal{G}f(m,\xi)|^2 = \left| \int_{-\infty}^{+\infty} f(t)g(t-m)e^{-i\xi t}dt \right|^2 \tag{A.1.7}$$

When using the windowed Fourier atoms, the Heisenberg box centered at (m,ξ) and of area $\sigma_t(m,\xi)\sigma_\omega(m,\xi)$ is independent of (m,ξ). [6]

In other words, a Gabor transform has the same resolution across the time frequency place as shown in the figure A.2

In practice, g has a compact support. Therefore, its Fourier transform $\hat{g}$ cannot be zero on a whole interval and has necessarily an infinite support, with a main lobe centered at $w = 0$ which decays to zero with oscillations.

As shown in the figure A.3, there are three characteristics that motivate the choice of the function g, they all characterize the energy spread of $\hat{g}$: The root mean-square bandwidth $\Delta\omega$ defined as the frequency at which we get $\frac{|\hat{g}(\frac{\Delta\omega}{2})|}{|\hat{g}(0)|} = \frac{1}{2}$, the maximum amplitude A of the first side lobes [7] and the polynomial exponent p which described the asymptotic decay of $|\hat{g}(.)|$. (i.e p such that $|\hat{g}(\omega)| = \underset{\omega\to+\infty}{O}(\omega^{-p-1})$).

6. Indeed, $\sigma_t^2(m,\xi) = \int_{-\infty}^{+\infty}(t-m)^2|g_{m,\xi}(t)|^2dt = \int_{-\infty}^{+\infty} t^2|g(t)|^2dt = \sigma_t^2$ and $\sigma_\omega^2(m,\xi) = \frac{1}{2\pi}\int_{-\infty}^{+\infty}(\omega-\xi)^2|\hat{g}_{m,\xi}(\omega)|^2d\omega = \frac{1}{2\pi}\int_{-\infty}^{+\infty}\omega^2|\hat{g}(\omega)|^2d\omega = \sigma_\omega^2$

7. Usually measured in decibels (a logarithmic scale)

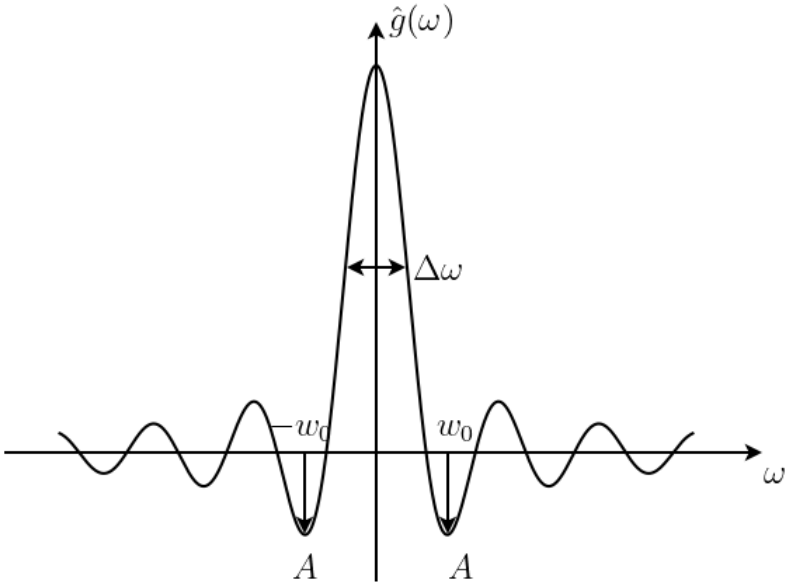


FIGURE A.3 – Characteristics of the energy spread of the Fourier transform of g

Ultimately, we would like to perform the Gabor transform on a discrete signal. Therefore, it is necessary to discretize both in time and in frequency.

Intuitively, we can expect that the discrete Gabor transform is complete if the Heisenberg boxes of all the atoms resulting from the discretization cover the time frequency plane.

The following table summarizes the three properties for the different window functions commonly used.

Name	$g(t)$	$\Delta\omega$	A	p
Hammming	$0.54 - 0.46\cos(2\pi t)$	8.54	-43 db	0
Gaussian	$\exp(-18t^2)$	9.73	-55 db	0
Hanning	$\cos^2(\pi t)$	9.04	-32 db	2
Blackman	$0.42 + 0.5\cos(2\pi t) + 0.08\cos(4\pi t)$	10.55	-58 db	2

In our approach, we have used the hamming function.

A.2 The FFT algorithm

The basic idea behind the FFT is that the DFT may be implemented much more efficiently when the number of data points N is a power of 2 using the following *divide and conquer* well known strategy.

In our implementation $N = 128 = 2^7$, Let V_{128} be the Vandermonde matrix in equation 3.3.1.

So, equation 3.3.1 can be written as follows :

$$\hat{f} = V_N f = \begin{bmatrix} I_{\frac{N}{2}} & D_{\frac{N}{2}} \\ I_{\frac{N}{2}} & -D_{\frac{N}{2}} \end{bmatrix} \begin{bmatrix} V_{\frac{N}{2}} & 0 \\ 0 & V_{\frac{N}{2}} \end{bmatrix} \begin{bmatrix} f_{\text{even}} \\ f_{\text{odd}} \end{bmatrix}$$

Where f_{even} are the even index elements of f, and similarly f_{odd} are the odd index elements of f. $I_{\frac{N}{2}}$ is the $\frac{N}{2} \times \frac{N}{2}$ identity matrix and $D_{\frac{N}{2}}$ is defined as follows :

$$D_{\frac{N}{2}} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & w_N & 0 & \dots & 0 \\ 0 & 0 & w_N^2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & w_N^{\frac{N}{2}-1} \end{bmatrix}$$

This factorization can be repeateed, and V_N can be represented by $V_{\frac{N}{2}}$, which can be represented by $V_{\frac{N}{2^2}}$, etc.

Therefore, the FFT involves an efficient interleaving of even and odd indices of subvectors of f, and the computation of several smaller 2×2 DFT computations.

A.3 Learning process for the conditional variational autoencoder

Let's compute the gradient of $\xi(\theta, \phi) := \sum_{i=1}^{N} \left(-\mathbb{E}_{q(z_i|x_i,y_i)} \left[\log(p_\theta(x_i|z_i, y_i)) \right] \right)$ w.r.t θ

$$\begin{aligned}
\nabla_\theta \xi(\theta, \phi) &= -\nabla_\theta \sum_{i=1}^{N} \left(\mathbb{E}_{q(z_i|x_i,y_i)} \left[\log(p_\theta(x_i|z_i, y_i)) \right] \right) \\
&= -\sum_{i=1}^{N} \left(\mathbb{E}_{q(z_i|x_i,y_i)} \left[\nabla_\theta \log(p_\theta(x_i|z_i, y_i)) \right] \right) \\
&\approx -\sum_{i=1}^{N} \left(\nabla_\theta \log(p_\theta(x_i|\widehat{z_i}, y_i)) \right) \quad \text{(Monte Carlo Approximation by sampling } \widehat{z_i} \sim q(z_i|x_i, y_i)) \\
&\approx -\frac{N}{M} \sum_{s=1}^{M} \left(\nabla_\theta \log(p_\theta(x_{i_s} \mid \widehat{z_{i_s}}, y_{i_s})) \right) \quad \text{(Where M is the batch size)}
\end{aligned}$$

Let's compute the gradient of $\xi(\theta, \phi)$ w.r.t ϕ by using the reparameterization trick. So, $\widehat{z_i} \sim q(z_i|x_i, y_i, \phi)$

is equivalent to $\widehat{z_i} = h(\varepsilon_i, x_i, y_i, \phi)$ where h is differentiable w.r.t. ϕ and $\varepsilon_i \sim p(\varepsilon_i) = \mathcal{N}(0, I)$

$$
\begin{aligned}
\nabla_\phi \xi(\theta, \phi) &= -\nabla_\phi \sum_{i=1}^{N} \left(\mathbb{E}_{q(z_i|x_i,y_i)} \left[\log(p_\theta(x_i|z_i, y_i)) \right] \right) \\
&= -\sum_{i=1}^{N} \nabla_\phi \left(\mathbb{E}_{q(z_i|x_i,y_i)} \left[\nabla_\theta \log(p_\theta(x_i|z_i, y_i)) \right] \right) \\
&= -\sum_{i=1}^{N} \nabla_\phi \left(\mathbb{E}_{p(\varepsilon_i)} \left[\nabla_\theta \log(p_\theta(x_i \mid h(\varepsilon_i, x_i, y_i, \phi))) \right] \right) \\
&= -\sum_{i=1}^{N} \int_{\varepsilon_i} \nabla_\phi \log(p_\theta(x_i \mid h(\varepsilon_i, x_i, y_i, \phi))) p(\varepsilon_i) d\varepsilon_i \\
&= -\sum_{i=1}^{N} \mathbb{E}_{p(\varepsilon_i)} \left[\nabla_\phi \log(p_\theta(x_i \mid h(\varepsilon_i, x_i, y_i, \phi))) \right] \\
&\approx -\sum_{i=1}^{N} \nabla_\phi \log(p_\theta(x_i \mid h(\widehat{\varepsilon_i}, x_i, y_i, \phi))) \quad \text{(Monte Carlo Approximation by sampling } \widehat{\varepsilon_i} \sim \mathcal{N}(0, I)) \\
&\approx -\frac{N}{M} \sum_{s=1}^{M} \nabla_\phi \log(p_\theta(x_{i_s} \mid h(\widehat{\varepsilon_{i_s}}, x_{i_s}, y_{i_s} \phi))) \quad \text{(using the batch sample)}
\end{aligned}
$$

We used TensorFlow to calculate the gradients $\nabla_\theta \log(p_\theta(x_{i_s} \mid \widehat{z_{i_s}}))$, $\nabla_\phi \log(p_\theta(x_{i_s} \mid h(\widehat{\varepsilon_{i_s}}, x_{i_s}, \phi)))$.

It is straightforward to calculate the gradient of the $\mathcal{KL}$ divergence term since the $\mathcal{KL}$ divergence between two gaussians is expressed analytically.

A.4 The Forward Backward Algorithm for calculating filtering and smoothing probabilities

A.4.1 Filtering probabilities : Forward Algorithm

In order to compute the filtering probabilities $\xi \in \mathbb{R}^{T \times M}$, we will take advantage of the conditional independencies in the graphical model, breaking the problem into pieces as follows :

1. **Expressing the Filtering probabilities in term of alpha variables :**

   First, let's introduce $\alpha \in \mathbb{R}^{T \times M}$:

$$
\forall (t, h) \in [\![1, T]\!] \times [\![1, M]\!] \quad \alpha(t, h) := p(\tilde{X}_1, \ldots, \tilde{X}_t, H_t = h) \tag{A.4.8}
$$

The filtering probabilities can be expressed using the α variables, which are easy to compute (recursively).

Indeed, for all $(t, h) \in [\![1, T]\!] \times [\![1, M]\!]$:

$$\begin{aligned}
\xi(t,h) &:= p(H_t = h | \tilde{X}_1, \dots, \tilde{X}_t) \\
&= \frac{p(H_t = h, \tilde{X}_1, \dots, \tilde{X}_t)}{p(\tilde{X}_1, \dots, \tilde{X}_t)} \\
&= \frac{p(H_t = h, \tilde{X}_1, \dots, \tilde{X}_t)}{\sum_{h'=1}^{M} p(H_t = h', \tilde{X}_1, \dots, \tilde{X}_t)} \\
&= \frac{\alpha(t,h)}{\sum_{h'=1}^{M} \alpha(t,h')}
\end{aligned}$$

So,

$$\boxed{\forall (t,h) \in [\![1,T]\!] \times [\![1,M]\!] \quad \xi(t,h) = \frac{\alpha(t,h)}{\sum_{h'=1}^{M} \alpha(t,h')}} \tag{A.4.9}$$

Notation :

$$\forall t \in [\![1,T]\!] \quad \alpha_t = (\alpha(t,1), \dots, \alpha(t,M))^T \quad \text{and} \quad \xi_t = (\xi(t,1), \dots, \xi(t,M))^T$$

The equation A.4.9 can then be written as follows :

$$\boxed{\forall t \in [\![1,T]\!] \quad \xi_t = \frac{\alpha_t}{\mathbb{1}_M^T \alpha_t}} \tag{A.4.10}$$

2. **Calculating the alpha variables recursively :**

Using the marginalization on the previous hidden state and exploiting the graphical model independencies, we obtain :

For all $(t, h) \in [\![1, T]\!] \times [\![1, M]\!]$:

$$
\begin{aligned}
\alpha(t,h) &= p(\tilde{X}_1,\dots,\tilde{X}_t, H_t = h) \\
&= \sum_{h'=1}^{M} p(\tilde{X}_1,\dots,\tilde{X}_t, H_t = h, H_{t-1} = h') \\
&= \sum_{h'=1}^{M} p(\tilde{X}_t|\tilde{X}_1,\dots,\tilde{X}_{t-1}, H_t = h, H_{t-1} = h') p(\tilde{X}_1,\dots,\tilde{X}_{t-1}, H_t = h, H_{t-1} = h') \\
&= \sum_{h'=1}^{M} p(\tilde{X}_t|H_t = h) p(H_t = h|\tilde{X}_1,\dots,\tilde{X}_{t-1}, H_{t-1} = h') p(\tilde{X}_1,\dots,\tilde{X}_{t-1}, H_{t-1} = h') \\
&= \sum_{h'=1}^{M} p(\tilde{X}_t|H_t = h) \underbrace{p(H_t = h|H_{t-1} = h')}_{Q_{hh'}} \underbrace{p(\tilde{X}_1,\dots,\tilde{X}_{t-1}, H_{t-1} = h')}_{\alpha(t-1,h')}
\end{aligned}
$$

So,

$$
\boxed{\forall (t,h) \in [\![1,T]\!] \times [\![1,M]\!] \quad \alpha(t,h) = \sum_{h'=1}^{M} p(\tilde{X}_t|H_t = h) Q_{hh'} \alpha(t-1,h')} \tag{A.4.11}
$$

Let's introduce the emission tensor $\Gamma \in [\![1,T]\!] \times [\![1,M]\!] \times [\![1,M]\!]$ such that :

$$
\forall t \in [\![1,T]\!] : \Gamma(t) = \begin{pmatrix} p(\tilde{X}_t|H_t = 1) & & \\ & \ddots & \\ & & p(\tilde{X}_t|H_t = M) \end{pmatrix}
$$

The equation A.4.11 can be written as follows :

$$
\boxed{\alpha_0 = \Gamma(0)\pi, \quad \forall t \geq 1 \; \alpha_t = \Gamma(t) Q^T \alpha_{t-1}} \tag{A.4.12}
$$

3. **Handling numerical issues :**

From the above sections, we conclude that, given an observed sequence, the filtering probabilities can be obtained by :

— Calculating the alpha variables recursively, as shown in the equations A.4.12.

— Deducing the filtering probabilities, as shown in equation A.4.10.

However, attention must be paid to numerical issues during the implementation. Since the recursion in A.4.12 implies several multiplications of small numbers, the numbers underflow quite rapidly.

To avoid underflow, it suffices to normalize. Indeed, normalizing the alpha variables (A.4.8), which means dividing $p(\tilde{X}_1, \dots, \tilde{X}_t, H_t = h)$ by $p(\tilde{X}_1, \dots, \tilde{X}_t)$ yields conditionals $p(H_t = h|\tilde{X}_1, \dots, \tilde{X}_t)$. These conditionals, which represent the filtered estimates of the states, scaled in a stable manner. In sum, one should compute directly the filtering probabilities instead of computing recursively the alpha variables and then deducing the filtering estimates.

Therefore, let's introduce $(\tilde{\xi}_t)_{1\leq t\leq T}$ as follows :

$$\boxed{\forall t \in [\![1, T]\!] \quad \tilde{\xi}_t := \Gamma(t)Q^T\xi_{t-1}} \tag{A.4.13}$$

And $(c_t)_{1\leq t\leq T}$ defined as follows :

$$\boxed{\forall t \in [\![1, T]\!] \quad c_t := \mathbb{1}^T\tilde{\xi}_t} \tag{A.4.14}$$

We have then :

$$\begin{aligned} c_t &:= \mathbb{1}^T\tilde{\xi}_t \\ &= \mathbb{1}_M^T\Gamma(t)Q^T\xi_{t-1} \\ &= \mathbb{1}_M^T\frac{\overbrace{\Gamma(t)Q^T\alpha_{t-1}}^{\alpha_t}}{\mathbb{1}_M^T\alpha_{t-1}} \quad \text{(from A.4.10 and A.4.12)} \\ &= \frac{\mathbb{1}_M^T\alpha_t}{\mathbb{1}_M^T\alpha_{t-1}} \end{aligned}$$

So,

$$\boxed{\forall t \in [\![1, T]\!] \quad c_t = \frac{\mathbb{1}_M^T\alpha_t}{\mathbb{1}_M^T\alpha_{t-1}}} \tag{A.4.15}$$

And we have :

$$\begin{aligned}
\xi_t &= \frac{1}{\mathbb{1}_M^T \alpha_t} \alpha_t \quad \text{(from A.4.10)} \\
&= \frac{1}{\mathbb{1}_M^T \alpha_t} \Gamma(t) Q^T \alpha_{t-1} \quad \text{(from A.4.12)} \\
&= \Gamma(t) Q^T \underbrace{\frac{\alpha_{t-1}}{\mathbb{1}_M^T \alpha_{t-1}}}_{\xi_{t-1}} \frac{\mathbb{1}_M^T \alpha_{t-1}}{\mathbb{1}_M^T \alpha_t} \\
&= \underbrace{\Gamma(t) Q^T \xi_{t-1}}_{:=\tilde{\xi}_t} \frac{\mathbb{1}_M^T \alpha_{t-1}}{\mathbb{1}_M^T \alpha_t} \\
&= \frac{\tilde{\xi}_t}{c_t} \quad \text{(from A.4.15)}
\end{aligned}$$

So,

$$\boxed{\forall t \in [\![1, T]\!] \quad \xi_t = \frac{\tilde{\xi}_t}{c_t}} \tag{A.4.16}$$

Finally, the recursion consists in the following steps :

— Initialization of c_1 and ξ_1

— We move from t to $t+1$ as follows :

$$\begin{aligned}
\xi_t &\xrightarrow{\text{eq}A.4.13} \tilde{\xi}_{t+1} \xrightarrow{\text{eq}A.4.14} c_{t+1} \\
(c_{t+1} \text{ and } \tilde{\xi}_{t+1}) &\xrightarrow{\text{eq}A.4.16} \xi_{t+1}
\end{aligned}$$

4. **The Forward Algorithm :**

From the previous section, we conclude the following algorithm, called **Forward Algorithm** :

Algorithm 7 Forward Algorithm

Input: Observations $\tilde{X}_1 \dots \tilde{X}_T$
Output: $(\xi_t)_{1 \le t \le T}$ (The filtering probabilties)

$c_1 \leftarrow \mathbb{1}^T \Gamma(1) \pi$
$\xi_1 \leftarrow \Gamma(1)\pi / c_1$
for $t \leftarrow 2, \dots, T$ **do**
    $\tilde{\xi}_t \leftarrow \Gamma(t) Q^T \xi_{t-1}$
    $c_t \leftarrow \mathbb{1}^T \tilde{\xi}_t$
    $\xi_t \leftarrow \tilde{\xi}_t / c_t$
end for=0

A.4.2 Smoothing probabilities : Forward Backward Algorithm

1. **Expressing the smoothing probabilities in term of alpha and beta variables :**

   First, let's introduce $\beta \in \mathbb{R}^{T\times M}$:

$$\forall (t,h) \in [\![1,T]\!] \times [\![1,M]\!] \quad \beta(t,h) := p(\tilde{X}_{t+1},\dots,\tilde{X}_T | H_t = h) \tag{A.4.17}$$

   The smoothing probabilities can be expressed using the alpha and beta variables, which we can calculate (recursively).

   Indeed, for all $(t,h) \in [\![1,T]\!] \times [\![1,M]\!]$:

$$\begin{aligned}
\psi(t,h) &:= p(H_t = h | \tilde{X}_1,\dots,\tilde{X}_T) \\
&= \frac{p(H_t = h, \tilde{X}_1,\dots,\tilde{X}_T)}{p(\tilde{X}_1,\dots,\tilde{X}_T)} \\
&= \frac{p(H_t = h, \tilde{X}_1,\dots,\tilde{X}_t)p(\tilde{X}_{t+1},\dots,\tilde{X}_T | H_t = h, \tilde{X}_1,\dots,\tilde{X}_t)}{\sum_{h'=1}^{M} p(H_t = h', \tilde{X}_1,\dots,\tilde{X}_T)} \\
&= \frac{p(H_t = h, \tilde{X}_1,\dots,\tilde{X}_t)p(\tilde{X}_{t+1},\dots,\tilde{X}_T | H_t = h)}{\sum_{h'=1}^{M} p(H_t = h', \tilde{X}_1,\dots,\tilde{X}_T)} \\
&= \frac{\alpha(t,h)\beta(t,h)}{\sum\limits_{h'=1}^{M} \alpha(t,h')}
\end{aligned}$$

   So,

$$\boxed{\forall (t,h) \in [\![1,T]\!] \times [\![1,M]\!] \quad \psi(t,h) = \frac{\alpha(t,h)\beta(t,h)}{\sum\limits_{h'=1}^{M} \alpha(t,h')}} \tag{A.4.18}$$

   We also have for all $(t,h,h') \in [\![1,T]\!] \times [\![1,M]\!] \times [\![1,M]\!]$:

$$
\begin{aligned}
\phi(t,h,h') &:= p(H_t = h, H_{t+1} = h' | \tilde{X}_1, \dots, \tilde{X}_T) \\
&= \frac{p(\tilde{X}_1, \dots, \tilde{X}_T | H_t = h, H_{t+1} = h') p(H_t = h, H_{t+1} = h')}{p(\tilde{X}_1, \dots, \tilde{X}_T)} \\
&= \frac{p(\tilde{X}_1, \dots, \tilde{X}_t | \tilde{X}_{t+1}, \dots, \tilde{X}_T, H_t = h, H_{t+1} = h') p(\tilde{X}_{t+1}, \dots, \tilde{X}_T | H_t = h, H_{t+1} = h') p(H_t = h, H_{t+1} = h')}{p(\tilde{X}_1, \dots, \tilde{X}_T)} \\
&= \frac{\overbrace{p(\tilde{X}_1, \dots, \tilde{X}_t | H_t = h)} p(\tilde{X}_{t+1}, \dots, \tilde{X}_T | H_{t+1} = h') p(H_{t+1} = h' | H_t = h) \overbrace{p(H_t = h)}}{p(\tilde{X}_1, \dots, \tilde{X}_T)} \\
&= \frac{\alpha(t,h) p(\tilde{X}_{t+1}, \dots, \tilde{X}_T | H_{t+1} = h') p(H_{t+1} = h' | H_t = h)}{p(\tilde{X}_1, \dots, \tilde{X}_T)} \\
&= \frac{\alpha(t,h) p(\tilde{X}_{t+1} | \tilde{X}_{t+2}, \dots, \tilde{X}_T, H_{t+1} = h') p(\tilde{X}_{t+2}, \dots, \tilde{X}_T | H_{t+1} = h') p(H_{t+1} = h' | H_t = h)}{p(\tilde{X}_1, \dots, \tilde{X}_T)} \\
&= \frac{\alpha(t,h) p(\tilde{X}_{t+1} | H_{t+1} = h') \overbrace{p(\tilde{X}_{t+2}, \dots, \tilde{X}_T | H_{t+1} = h')}^{:= \beta(t+1,h')} \overbrace{p(H_{t+1} = h' | H_t = h)}^{Q_{hh'}}}{\sum_{h''=1}^{M} p(\tilde{X}_1, \dots, \tilde{X}_T, H_T = h'')} \\
&= \frac{\alpha(t,h) p(\tilde{X}_{t+1} | H_{t+1} = h') \beta(t+1,h') Q_{hh'}}{\sum_{h''=1}^{M} \alpha(T,h'')}
\end{aligned}
$$

So,

$$
\boxed{\forall (t,h,h') \in [\![1,T]\!] \times [\![1,M]\!] \times [\![1,M]\!] \quad \phi(t,h,h') = \frac{\alpha(t,h) p(\tilde{X}_{t+1} | H_{t+1} = h') \beta(t+1,h') Q_{hh'}}{\sum_{h''=1}^{M} \alpha(T,h'')}} \tag{A.4.19}
$$

Notation :

$$
\forall t \in [\![1,T]\!] \quad \beta_t = (\beta(t,1), \dots, \beta(t,M))^T \quad \phi_t = [\phi(t,h,h')]_{h,h'} \quad \psi_t = [\psi(t,h,h')]_{h,h'}
$$

So, The equation A.4.19 can then be written as :

$$
\boxed{\forall t \in [\![1,T]\!] \quad \phi_t = \frac{\text{diag}(\alpha_t) Q \Gamma(t+1) \text{diag}(\beta_{t+1})}{\mathbb{1}_M^T \alpha_T}} \tag{A.4.20}
$$

2. **Calculating the beta variables recursively :**

Using the marginalization on the next hidden state and exploiting the graphical model independencies, we obtain :

For all $(t,h) \in [\![1,T]\!] \times [\![1,M]\!]$:

$$\begin{aligned}
\beta(t,h) &= p(\tilde{X}_{t+1},\dots,\tilde{X}_T|H_t=h) \\
&= \sum_{h'=1}^{M} p(\tilde{X}_{t+1},\dots,\tilde{X}_T,H_{t+1}=h'|H_t=h) \\
&= \sum_{h'=1}^{M} p(\tilde{X}_{t+1},\dots,\tilde{X}_T|H_{t+1}=h',H_t=h)p(H_{t+1}=h'|H_t=h) \\
&= \sum_{h'=1}^{M} p(\tilde{X}_{t+1},\dots,\tilde{X}_T|H_{t+1}=h')p(H_{t+1}=h'|H_t=h) \\
&= \sum_{h'=1}^{M} p(\tilde{X}_{t+1}|\tilde{X}_{t+2},\dots,\tilde{X}_T,H_{t+1}=h')p(\tilde{X}_{t+2},\dots,\tilde{X}_T|H_{t+1}=h')p(H_{t+1}=h'|H_t=h) \\
&= \sum_{h'=1}^{M} p(\tilde{X}_{t+1}|H_{t+1}=h')p(\tilde{X}_{t+2},\dots,\tilde{X}_T|H_{t+1}=h')p(H_{t+1}=h'|H_t=h) \\
&= \sum_{h'=1}^{M} p(\tilde{X}_{t+1}|H_{t+1}=h')\beta(t+1,h')Q_{hh'}
\end{aligned}$$

So,

$$\boxed{\forall (t,h) \in [\![1,T]\!] \times [\![1,M]\!] \quad \beta(t,h) = \sum_{h'=1}^{M} p(\tilde{X}_{t+1}|H_{t+1}=h')\beta(t+1,h')Q_{hh'}} \tag{A.4.21}$$

The equation A.4.21 can be written as follows :

$$\boxed{\beta_T = \mathbb{1}_M, \quad \forall t \leq T-1 \ \beta_t = Q\Gamma(t+1)\beta_{t+1}} \tag{A.4.22}$$

A.5 M-step for HMM

We want to maximize : $\mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(p_\theta(\mathbf{h},\tilde{\mathbf{x}}))]$ with respect to $\theta = (\pi, Q, \mu, \Sigma)$

We have :

$$\begin{aligned}\mathbb{E}_{\mathbf{H}|\tilde{\mathbf{x}}}[\log(p_\theta(\mathbf{h},\tilde{\mathbf{x}}))] = &\sum_{h=1}^{M}\log(\pi_h)\psi(1,h) + \sum_{t=1}^{T-1}\sum_{h=1}^{M}\sum_{h'=1}^{M}\log(Q_{hh'})\phi(t,h,h') \\ &+ \sum_{t=1}^{T-1}\sum_{h=1}^{M}\log(\mathcal{N}(\tilde{X}_t;\mu_h,\Sigma_h))\psi(t,h)\end{aligned} \tag{A.5.23}$$

So, the optimization problem can be decomposed in three sub-problems.

$$\pi^{(i+1)} = \arg\min_{\pi} -\sum_{h=1}^{M}\log(\pi_h)\psi(1,h) \quad \text{such that} \quad \pi \perp \mathbb{1}_{\mathbb{M}} \tag{A.5.24}$$

$$Q^{(i+1)} = \arg\min_{Q} -\sum_{t=1}^{T-1}\sum_{h=1}^{M}\sum_{h'=1}^{M}\log(Q_{hh'})\phi(t,h,h') \quad \text{such that} \quad \sum_{h'=1}^{M} Q_{hh'} = 1 \tag{A.5.25}$$

$$(\mu^{(i+1)},\Sigma^{(i+1)}) = \arg\min_{(\mu,\Sigma)} -\sum_{t=1}^{T-1}\sum_{h=1}^{M}\log(\mathcal{N}(\tilde{X}_t;\mu_h,\Sigma_h))\psi(t,h) \tag{A.5.26}$$

A.5.1 Update of the initial state

We use the Langrangian to solve the first optimization problem A.5.24 :

$$\mathcal{L}(\pi,\lambda) = -\sum_{h=1}^{M}\log(\pi_h)\psi(1,h) + \lambda\left(\sum_{h=1}^{M}\pi_h - 1\right) \tag{A.5.27}$$

We have :

$$\forall h \in [\![1,M]\!] \quad 0 = \frac{\partial\mathcal{L}}{\partial\pi_h}(\pi^{(i+1)},\lambda) = -\frac{\psi(1,h)}{\pi_h^{(i+1)}} + \lambda \tag{A.5.28}$$

Thus,

$$\forall h \in [\![1,M]\!] \quad \pi_h^{(i+1)} = \frac{\psi(1,h)}{\lambda} \tag{A.5.29}$$

And from :

$$1 = \sum_{h=1}^{M}\pi_h^{(i+1)} = \sum_{h=1}^{M}\frac{\psi(1,h)}{\lambda} = \frac{1}{\lambda}\underbrace{\sum_{h=1}^{M}\psi(1,h)}_{=1} \tag{A.5.30}$$

We get :

$$\lambda = 1 \tag{A.5.31}$$

Therefore,

$$\boxed{\forall h \in [\![1, M]\!] \quad \pi_h^{(i+1)} = \psi(1, h)} \tag{A.5.32}$$

A.5.2 Update of the transition matrix

Again, we use the Langrangian to solve the second optimization problem A.5.25 :

$$\mathcal{L}(Q, \lambda_1, \ldots, \lambda_M) = -\sum_{t=1}^{T-1}\sum_{h=1}^{M}\sum_{h'=1}^{M} \log(Q_{hh'})\phi(t, h, h') + \sum_{h=1}^{M} \lambda_h \left(\sum_{h'=1}^{M} Q_{hh'} - 1 \right) \tag{A.5.33}$$

We have :

$$\forall (h, h') \in [\![1, M]\!] \times [\![1, M]\!] \quad 0 = \frac{\partial \mathcal{L}}{\partial Q_{hh'}}(Q^{(i+1)}, \lambda) = -\sum_{t=1}^{T-1} \frac{\phi(t, h, h')}{Q_{hh'}^{(i+1)}} + \lambda_h \tag{A.5.34}$$

Thus,

$$\forall (h, h') \in [\![1, M]\!] \times [\![1, M]\!] \quad Q_{hh'}^{(i+1)} = \frac{\sum_{t=1}^{T-1} \phi(t, h, h')}{\lambda_h} \tag{A.5.35}$$

And from :

$$\forall h \in [\![1, M]\!] \quad 1 = \sum_{h'=1}^{M} Q_{hh'}^{(i+1)} = \sum_{h'=1}^{M} \left(\frac{\sum_{t=1}^{T-1} \phi(t, h, h')}{\lambda_h} \right) = \frac{1}{\lambda_h} \sum_{t=1}^{T-1} \underbrace{\sum_{h'=1}^{M} \phi(t, h, h')}_{=\psi(t,h)} \tag{A.5.36}$$

We get :

$$\forall h \in [\![1, M]\!] \quad \lambda_h = \sum_{t=1}^{T-1} \psi(t, h) \tag{A.5.37}$$

Therefore,

$$\boxed{\forall (h,h') \in [\![1,M]\!] \times [\![1,M]\!] \quad Q_{hh'}^{(i+1)} = \frac{\sum_{t=1}^{T-1} \phi(t,h,h')}{\sum_{t=1}^{T-1} \psi(t,h)}} \tag{A.5.38}$$

A.5.3 Update of the emission distribution

We have :

$$(\mu^{(i+1)}, \Sigma^{(i+1)}) = \underset{(\mu,\Sigma)}{\arg\min} \underbrace{-\sum_{t=1}^{T-1}\sum_{h=1}^{M} \log(\mathcal{N}(\tilde{X}_t; \mu_h, \Sigma_h))\psi(t,h)}_{J(\mu,\Sigma)} \tag{A.5.39}$$

1. **Update** μ :

   Let's fix $h \in [\![1,M]\!]$ and $t \in [\![1,T]\!]$

   We have :

$$\log(\mathcal{N}(\tilde{X}_t; \mu_h, \Sigma_h)) = \frac{D}{2}\log(2\pi) - \frac{1}{2}\log(\det(\Sigma_h)) - \frac{1}{2}(\tilde{X}_t - \mu_h)^T \Sigma_h^{-1} (\tilde{X}_t - \mu_h) \tag{A.5.40}$$

   We define

$$\xi : \mu_h \overset{f}{\mapsto} \mu_h - \tilde{X}_t \overset{g}{\mapsto} (\tilde{X}_t - \mu_h)^T \Sigma_h^{-1} (\tilde{X}_t - \mu_h) \tag{A.5.41}$$

   Obviously,

$$\nabla f(x) = x \tag{A.5.42}$$

   We have

$$\begin{aligned}\forall(x,\epsilon)\in\mathbb{R}^D\times\mathbb{R}^D \quad g(x+\epsilon)-g(x) &= (x+\epsilon)^T\Sigma_h^{-1}(x+\epsilon)-x^T\Sigma_h^{-1}x\\ &= \epsilon^T\Sigma_h^{-1}x+x^T\Sigma_h^{-1}\epsilon+\epsilon^T\Sigma_h^{-1}\epsilon\\ &= \epsilon^T(\Sigma_h^{-1}x+(\Sigma_h^{-1})^Tx)+o(||\epsilon||)\\ &= \epsilon^T(2\Sigma_h^{-1}x)+o(||\epsilon||)\\ &= \underbrace{\langle 2\Sigma_h^{-1}x),\epsilon\rangle}_{dg_x(\epsilon)}+o(||\epsilon||)\\ &= \langle\nabla g(x),\epsilon\rangle\end{aligned}$$

So,

$$\nabla g(x) = 2\Sigma_h^{-1}x \tag{A.5.43}$$

Therefore,

$$\begin{aligned}\forall x\in\mathbb{R}^D \quad \langle\nabla\xi(\mu_h),x\rangle &= d(\xi)_{\mu_h}(x)\\ &= d(g\circ f)_{\mu_h}(x)\\ &= d(g)_{\mu_h-\tilde{X}_t}\left(d(f)_{\mu_h}(x)\right)\\ &= d(g)_{\mu_h-\tilde{X}_t}\left(\langle\nabla f(\mu_h),x\rangle\right)\\ &= d(g)_{\mu_h-\tilde{X}_t}(x)\\ &= \langle\nabla g(\mu_h-\tilde{X}_t),x\rangle\\ &= \langle 2\Sigma_h^{-1}(\mu_h-\tilde{X}_t),x\rangle\end{aligned}$$

So,

$$\nabla\xi(\mu_h) = 2\Sigma_h^{-1}(\mu_h-\tilde{X}_t) \tag{A.5.44}$$

Thus,

$$0 = \nabla_{\mu_h}J(\mu_h^{(i+1)},\Sigma) = -\sum_{t=1}^{T}\psi(t,h)\Sigma_h^{-1}(\mu_h-\tilde{X}_t) \tag{A.5.45}$$

Therefore,

$$\boxed{\forall h \in [\![1, M]\!] \quad \mu_h^{(i+1)} = \frac{\sum_{t=1}^{T} \psi(t,h)\tilde{X}_t}{\sum_{t=1}^{T} \psi(t,h)}} \tag{A.5.46}$$

2. **Update** Σ :

Let's denote $\Omega = (\Omega_h)_{h\in[\![1,M]\!]} = (\Sigma_h^{-1})_{h\in[\![1,M]\!]}$

So J becomes a function of μ and Ω :

$$J(\mu, \Omega) = -\sum_{t=1}^{T-1}\sum_{h=1}^{M} \log(\mathcal{N}(\tilde{X}_t; \mu_h, \Omega_h))\psi(t,h) \tag{A.5.47}$$

We also have for $h \in [\![1, M]\!]$ and $t \in [\![1, T]\!]$:

$$\log(\mathcal{N}(\tilde{X}_t; \mu_h, \Omega_h)) = \frac{D}{2}\log(2\pi) - \frac{1}{2}\log(\det(\Omega_h)) - \frac{1}{2}(\tilde{X}_t - \mu_h)^T\Omega_h(\tilde{X}_t - \mu_h) \tag{A.5.48}$$

Thus, for $h \in [\![1, M]\!]$:

$$\nabla_{\Omega_h} J(\mu, \Omega) = \nabla_{\Omega_h}\left(-\sum_{t=1}^{T-1} \log(\mathcal{N}(\tilde{X}_t; \mu_h, \Omega_h))\psi(t,h)\right) \tag{A.5.49}$$

$$= -\sum_{t=1}^{T-1}\psi(t,h)\left(-\frac{1}{2}\nabla_{\Omega_h}\underbrace{\log(\det(\Omega_h)}_{:=u(\Omega_h)} + \frac{1}{2}\nabla_{\Omega_h}\underbrace{(\tilde{X}_t - \mu_h)^T\Omega_h(\tilde{X}_t - \mu_h)}_{:=v(\Omega_h)}\right) \tag{A.5.50}$$

Let's calculate $\nabla_{\Omega_h} u(\Omega_h)$

$$\begin{aligned}
\forall H \in \mathbb{R}^{D\times D} \quad u(\Omega_h + H) - u(\Omega_h) &= \log(\det(\Omega_h + H)) - \log(\det(\Omega_h)) \\
&= \log(\det(\Omega_h^{\frac{1}{2}}(I_D + \Omega_h^{\frac{-1}{2}} H \Omega_h^{\frac{-1}{2}})\Omega_h^{\frac{1}{2}})) - \log(\det(\Omega_h)) \\
&= \log(\det(\Omega_h)) + \log(\det(I_D + \Omega_h^{\frac{-1}{2}} H \Omega_h^{\frac{-1}{2}})) - \log(\det(\Omega_h)) \\
&= \log(\det(I_D + \Omega_h^{\frac{-1}{2}} H \Omega_h^{\frac{-1}{2}})) \qquad \text{(A.5.51)} \\
& \qquad \text{(A.5.52)}
\end{aligned}$$

We can decompose $\Omega_h^{\frac{-1}{2}} H \Omega_h^{\frac{-1}{2}}$ as follows :

$$\Omega_h^{\frac{-1}{2}} H \Omega_h^{\frac{-1}{2}} = U \begin{pmatrix} \omega_1 & & \\ & \ddots & \\ & & \omega_D \end{pmatrix} U^T$$

The equation A.5.51 becomes :

$$\forall H \in \mathbb{R}^{D\times D} \quad u(\Omega_h + H) - u(\Omega_h) = \sum_{i=1}^{D} \log(1+\omega_i) \tag{A.5.53}$$

$$= \sum_{i=1}^{D} \omega_i + o(||H||) \tag{A.5.54}$$

$$= \text{tr}(\Omega_h^{\frac{-1}{2}} H \Omega_h^{\frac{-1}{2}}) + o(||H||) \tag{A.5.55}$$

$$= \underbrace{\text{tr}(\Omega_h^{-1} H)}_{du_{\Omega_h}(H)} + o(||H||) \tag{A.5.56}$$

$$= \langle \nabla_{\Omega_h} u(\Omega_h), H \rangle + o(||H||) \tag{A.5.57}$$

$$\tag{A.5.58}$$

Therefore,

$$\nabla_{\Omega_h} u(\Omega_h) = \Omega_h^{-1} = \Sigma_h \tag{A.5.59}$$

Let's calculate $\nabla_{\Omega_h} v(\Omega_h)$. We have :

$$\forall H \in \mathbb{R}^{D\times D} \quad v(\Omega_h + H) - v(\Omega_h) = (\tilde{X}_t - \mu_h)^T (\Omega_h + H)(\tilde{X}_t - \mu_h) - (\tilde{X}_t - \mu_h)^T \Omega_h (\tilde{X}_t - \mu_h)$$

$$\tag{A.5.60}$$

$$= \text{tr}((\tilde{X}_t - \mu_h)^T (\Omega_h + H)(\tilde{X}_t - \mu_h)) - \text{tr}((\tilde{X}_t - \mu_h)^T \Omega_h (\tilde{X}_t - \mu_h)) \tag{A.5.61}$$

$$= \text{tr}(H(\tilde{X}_t - \mu_h)^T (\tilde{X}_t - \mu_h)) \tag{A.5.62}$$

$$= \text{tr}((\tilde{X}_t - \mu_h)^T (\tilde{X}_t - \mu_h) H) \tag{A.5.63}$$

$$= \underbrace{\text{tr}(((\tilde{X}_t - \mu_h)(\tilde{X}_t - \mu_h)^T)^T H)}_{dv_{\Omega_h}(H)} \tag{A.5.64}$$

$$= \langle \nabla_{\Omega_h} v(\Omega_h), H \rangle \tag{A.5.65}$$

$$\tag{A.5.66}$$

Therefore,

$$\nabla_{\Omega_h} v(\Omega_h) = (\tilde{X}_t - \mu_h)(\tilde{X}_t - \mu_h)^T \tag{A.5.67}$$

From the equation A.5.50, A.5.59 and A.5.67, we conclude :

$$\nabla_{\Omega_h} J(\mu, \Omega) = -\sum_{t=1}^{T-1} \psi(t,h) \left(-\frac{1}{2}\Sigma_h + \frac{1}{2}(\tilde{X}_t - \mu_h)(\tilde{X}_t - \mu_h)^T \right) \tag{A.5.68}$$

We set the gradient with respect to Σ to zero, we obtain,

$$0 = \nabla_{\mu_h} J(\mu, \Sigma_h^{(i+1)}) = -\sum_{t=1}^{T-1} \psi(t,h) \left(-\frac{1}{2}\Sigma_h^{(i+1)} + \frac{1}{2}(\tilde{X}_t - \mu_h)(\tilde{X}_t - \mu_h)^T \right) \tag{A.5.69}$$

Finally,

$$\boxed{\forall h \in [\![1, M]\!] \quad \Sigma_h^{(i+1)} = \frac{\sum_{t=1}^{T} \psi(t,h)(\tilde{X}_t - \mu_h)(\tilde{X}_t - \mu_h)^T}{\sum_{t=1}^{T} \psi(t,h)}} \tag{A.5.70}$$

B | Bayesian Approach with Attention Mechanism for Learning to Rank Financial Assets

B.1 Ranking Measures

We refer the reader to (Liu et al., 2009) for more information about the subject. Because there are a large number of possible ranking models, a mechanism for evaluating them is required. We present there two evaluation measures, the NDCG score and the Mean Average Precision. There exist other evaluation metrics, such as Mean Reciprocal Rank (Radev, Qi, Wu, & Fan, 2002), weighted Kendall' τ (Kendall & Gibbons, 1990) that we do not develop here. These evaluation measures are computed at the query level and are position based. Rank positions are explicitly used, so optimizing directly these measures is quite difficult due to the non-differentiability of the ranking operator. However, some authors have attempted to overcome this problem. refer for instance to (Valizadegan, Jin, Zhang, & Mao, 2009) or to (Swezey, Grover, Charron, & Ermon, 2021).

B.1.1 NDCG

Using the the feature vector $x^{(i)}$ and the corresponding score vector y_i for $1 \leq i \leq m$, we define the Discounted Cumulative Gain, given a ranking function $f_\theta(.)$, as the following :

$$DCG(f_\theta(x^{(i)}), y_i) = \sum_{j=1}^{n^{(i)}} \frac{G(y_{ij})}{\log(1 + r_j^{(i)})}$$

where $r_j^{(i)}$ is the predicted rank of $x_j^{(i)}$ using $f_\theta(.)$ and $G(.)$ is a function that can be defined as $G(y_{ij}) = 2^{y_{ij}} - 1$ or as $G(y_{ij}) = y_{ij}$ depending on sources in literature.. We take the first form in our experimental setting.

There exist a normalized version, known as the Normalized Discounted Cumulative Gain, defined as the following :

$$NDCG(f_\theta(x^{(i)}), y_i) = \frac{1}{Z^{(i)}} \sum_{j=1}^{n^{(i)}} \frac{G(y_{ij})}{\log(1 + r_j^{(i)})}$$

where $Z^{(i)} = DCG(\hat{f}_\theta(x^{(i)}), y_i)$ with $\hat{f}_\theta(x^{(i)})$ the best ranking function, i.e, the ranking function that rank the feature according to the rank of y_i. Hence, the NDCG computes a score in the closed interval

$[0, 1]$.

Finally, we refer to $\overline{NDCG}(f_\theta(x^{(i)}), y_i)$ to the mean metric, i.e :

$$\overline{NDCG}(f_\theta(.), y) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{Z^{(i)}} \sum_{j=1}^{n^{(i)}} \frac{G(y_{ij})}{\log(1 + r_j^{(i)})}$$

where $y = (y^{(1)}, \cdots, y^{(m)})$.

For theoretical support, we refer to (Y. Wang, Wang, Li, He, & Liu, 2013). An analysis of the behavior of NDCG as the number of objects to rank getting large is presented and some properties that make NDCG a good ranking measure are given. On the link between NDCG and ListNet we refer to (Bruch, Wang, Bendersky, & Najork, 2019)[Theorem 3]. Based on binary scores, it is shown that the $\log(\bar{[}NDCG])$ represents an upper bound on the ListNet Loss function over the entire dataset. For non binary scores, (Bruch, 2021) proposes a modification to ListNet to keep the upper bound.

B.1.2 Kendall measure

Kendall'τ (Kendall & Gibbons, 1990) is a measure of the correspondence between two rankings. It also known as the Kendall Rank Correlation Coefficient. It corresponds to the number of pairwise disagreements between the two input full rankings of objects. Let consider $\tilde{y}_{ij}$ is the rank of y_{ij} and $r_j^{(i)}$ is the predicted rank of $x_j^{(i)}$ for $1 \leq j \leq n^{(i)}$, then :

$$\text{Kendall } \tau(f_\theta(x^{(i)}), y_i) \quad = \quad \sum_{k=1}^{n^{(i)}} \sum_{l=1}^{n^{(i)}} \frac{\mathbb{1}_{\left((\tilde{y}_{ik} - \tilde{y}_{il})(r_k^{(i)} - r_l^{(i)}) < 0\right)}}{\binom{n^{(i)}}{2}}$$

Kendall'τ can be viewed as the minimum number of interchanges of adjacent pairs of the objects that is necessary in order to transform the ordering of objects according to y_i into the ordering of objects according to $f_\theta(x^{(i)})$. The normalization factor represents the worst case scenario where the order of objects based on y_i is the opposite of the order of objects based on $f_\theta(x^{(i)})$.

B.2 Exhaustive Results

	Cluster	*Annualized Return*	*Annualized Volatility*	*Sharpe Ratio*	*MDD*	*Sortino Ratio*	*Calmar Ratio*	*% Daily Positive Returns*
Cross Sectional Momentum (T=126)	A	15,91	18,91	0,84	-34,50	1,02	0,46	56,03
	B	14,12	19,68	0,72	-39,76	0,84	0,36	54,60
	C	13,55	28,32	0,48	-49,37	0,63	0,27	51,4
Cross Sectional Momentum (T=252)	A	15,49	19,11	0,81	-33,22	0,98	0,47	56,20
	B	13,88	19,67	0.71	-40,19	0,83	0,35	55,00
	C	13,30	25,76	0,52	-46,30	0,66	0,29	53,29
Ranking Momentum	A	16,25	20,29	0,80	-40,28	0,94	0,40	55,63
	B	14,7	19,73	0,75	-38,47	0,89	0,38	54,37
	C	9,85	21,65	0,46	-45,35	0,55	0,22	52,43
Proposed Model (RankCosine)	A	18,96	20,78	0,91	-38,04	1,08	0,50	55,17
	B	13,79	19,87	0,69	-40,16	0,82	0,34	54,43
	C	10,10	20,99	0,48	-42,58	0,59	0,24	53,06
Proposed Model (ListNet)	A	18,73	20,66	0,91	-37,76	1,08	0,50	55,63
	B	13,92	20,00	0,70	-40,45	0,82	0,34	53,80
	C	10,62	21,39	0,61	-43,34	0,61	0,25	53,00
Uniform		14,61	19,8	0,74	-39,76	0,87	0,37	54,72

TABLE B.1 – Comparing Results with A, B, C Clustering

	Cluster	*Annualized Return*	*Annualized Volatility*	*Sharpe Ratio*	*MDD*	*Sortino Ratio*	*Calmar Ratio*	*% Daily Positive Returns*
Cross Sectional Momentum (T=126)	$A+$	13,28	20,19	0,66	-32,93	0,82	0,40	54,49
	$A-$	18,05	18,74	0,96	-35,79	1,15	0,50	57,12
	B	14,12	19,68	0,72	-39,76	0,84	0,36	54,60
	$C+$	13,50	24,00	0,56	-45,82	0,69	0,29	54,43
	$C-$	13,55	28,32	0,48	-49,37	0,63	0,27	51,4
Cross Sectional Momentum (T=252)	$A+$	12,14	20,19	0,60	-32,91	0,73	0,37	55,69
	$A-$	19,01	19,04	0.99	-33,18	1,21	0,57	56,66
	B	13,88	19,67	0.71	-40,19	0,83	0,35	55,00
	$C+$	14,51	24,33	0,6	-41,78	0,76	0,35	52,32
	$C-$	10,06	29,16	0,35	-50,49	0,46	0,20	51,52
Ranking Momentum	$A+$	17,18	21,29	0,81	-39,12	0,99	0,44	56,20
	$A-$	16,21	20,45	0,79	-41,31	0,93	0,39	54,83
	B	14,7	19,73	0,75	-38,47	0,89	0,38	54,37
	$C+$	12,69	21,59	0,59	-38,11	0,74	0,33	52,37
	$C-$	6,21	23,06	0,27	-53,16	0,32	0,12	52,66
Proposed Model (RankCosine)	$A+$	21,23	21,38	0,99	-39,47	1,18	0,54	57,18
	$A-$	17,25	20,93	0,82	-37,89	0,99	0,46	54,55
	B	13,79	19,87	0,69	-40,16	0,82	0,34	54,43
	$C+$	10,33	20,87	0,49	-42,72	0,60	0,24	53,34
	$C-$	9,01	23,44	0,38	-44,59	0,50	0,20	51,97
Proposed Model (ListNet)	$A+$	20,45	21,57	0,95	-39,36	1,14	0,52	57,18
	$A-$	17,51	20,64	0,85	-36,95	1,02	0,47	55,06
	B	13,92	20,00	0,70	-40,45	0,82	0,34	53,80
	$C+$	11,13	21,47	0,52	-42,67	0,64	0,26	53,74
	$C-$	9,73	23,43	0,42	-44,59	0,53	0,22	51,74
Uniform		14,61	19,8	0,74	-39,76	0,87	0,37	54,72

TABLE B.2 – Comparing Results with Detailed Clusters

B.3 Volatility control

Modelize volatility of assets is possible using a plenty of different techniques. We refer to (Andersen, Bollerslev, Christoffersen, & Diebold, 2005) for an exhaustive survey of them. We focus on the Exponentially Weighted Moving Average methods as described in (Longerstaey & Spencer, 1996). Let $(r_t)_{1\leq t\leq T}$ the sequence of portfolio return at different time $1 \leq t \leq T$, then the estimate volatility of the portfolio at time t is given by the following quantity :

$$\hat{\sigma}_t^2 = (1-\lambda) r_{t-1}^2 + \lambda \hat{\sigma}_{t-1}^2$$

where $\lambda \in [0,1]$ and is known as the decay constant. (Longerstaey & Spencer, 1996) give different value of decay given the frequency of observation. For daily data, they found $\lambda = 0.94$ to be relevant and we fix it to the same value.

The weight w_{t+1} associated to volatility controlled portfolio at time $t+1$ is then :

$$w_{t+1} \quad = \quad \min\left(\frac{\sigma^{\text{target}}}{\hat{\sigma}_t}, w_{\text{max}}\right)$$

where σ^{target} is the target volatility and w_{max} is the maximum allowable exposure to the portfolio. We fix $\sigma^{\text{target}} = 25\%$ meaning that we target an annualised volatility of 25% and $w_{\text{max}} = 1$ (no leverage).

	Cluster	*Annualized Return*	*Annualized Volatility*	*Sharpe Ratio*	*MDD*	*Sortino Ratio*	*Calmar Ratio*	*% Daily Positive Returns*
Cross Sectional Momentum (T=126)	$A+$	10,80	16,32	0,66	-19,68	0,88	0,55	54,49
	$A-$	16,30	15,19	1,07	-21,31	1,36	0,76	57,12
	B	12,31	14,68	0,84	-23,12	1,08	0,53	54,60
	$C+$	8,38	16,32	0,51	-26,52	0,71	0,32	54,43
	$C-$	9,17	17,90	0,51	-30,45	0,76	0,30	51,4
Cross Sectional Momentum (T=252)	$A+$	10,66	16,24	0,66	-20,35	0,84	0,52	55,69
	$A-$	17,61	15,60	1,13	-21,46	1,45	0,82	56,66
	B	11,98	14,59	0,82	-23,45	1,06	0,54	55,00
	$C+$	7,83	16,66	0,47	-24,24	0,64	0,32	52,32
	$C-$	6,31	17,73	0,36	-34,39	0,53	0,18	51,52
Ranking Momentum	$A+$	14,24	16,22	0,88	-22,11	1,17	0,64	56,20
	$A-$	14,35	15,63	0,92	-23,15	1,18	0,62	54,83
	B	12,15	14,73	0,83	-22,69	1,07	0,54	54,37
	$C+$	9,64	15,72	0,61	-29,16	0,85	0,33	52,37
	$C-$	7,38	16,47	0,45	-32,89	0,61	0,22	52,66
Proposed Model (RankCosine)	$A+$	16,93	16,13	1,05	-23,14	1,35	0,73	57,18
	$A-$	14,21	15,67	0,91	-21,23	1,17	0,67	54,55
	B	11,74	14,80	0,79	-23,43	1,03	0,50	54,43
	$C+$	8,48	15,56	0,54	-30,70	0,73	0,28	53,34
	$C-$	6,07	16,70	0,36	-27,77	0,52	0,22	51,97
Proposed Model (ListNet)	$A+$	16,14	16,27	0,99	-23,44	1,28	0,69	57,18
	$A-$	14,52	15,50	0,94	-21,65	1,21	0,67	55,06
	B	11,69	14,88	0,79	-23,75	1,02	0,49	53,80
	$C+$	8,99	15,73	0,57	-28,19	0,77	0,32	53,74
	$C-$	6,81	16,73	0,41	-27,79	0,58	0,24	51,74
Uniform		12,50	14,74	0,85	-23,06	1,09	0,54	54,72

TABLE B.3 – Comparing Results with Detailed Clusters with control of volatility

Using the volatility control, the results of our proposed model are leveraged with a Cluster $A+$ and Cluster $C-$ that reflects expected behavior, which is Cluster $A+$ with good results and Cluster $C-$ with poor results. In terms of returns, Sharpe ratio, Sortino ratio and Calmar ratio, Cluster $A+$ and Cluster $C-$ are the best clusters with our proposed model. Furthermore, our proposed model provides the highest percentage of positive return for Cluster $A+$.

B.4 Assets Data

We choose to train and test our model for each year between 2015 and 2021 by defining a new universe each year, and we limit to assets with a minimum of five years of history to train our model efficiently. Therefore, we have an universe that we develop here for every year between 2015 and 2021. As a final clarification, we focus on stocks that belong to S&P 500 Industries Indices.

B.4.1 2015

Sectors - S&P 500 Sectorial Indices

Health Care, Utilities, Industrials, Consumer Discretionary, Financials, Information Technology, Consumer Staples, Materials, Communication Services, Energy

Industries - S&P 500 Industries Indices

Multiline Retail, Consumer Finance, Health Care Equipment Supplies, Internet Direct Marketing Retail, Personal Products, Chemicals, Machinery, Textiles Apparel Luxury Goods, Household Products, Health Care Providers Services, Air Freight Logistics, Capital Markets, Software, Banks, Metals Mining, Building Products, Trading Companies Distributors, Technology Hardware Storage Peripherals, Multi-Utilities, Industrial Conglomerates, Specialty Retail, Life Sciences Tools Services, Construction Engineering, Road Rail, Automobiles, Aerospace Defense, Energy Equipment Services, Electric Utilities, Leisure Products, Semiconductors Semiconductor Equipment, Beverages, Airlines, Communications Equipment, Food Products, IT Services, Diversified Telecommunication Services, Tobacco, Oil Gas Consumable Fuels, Electronic Equipment Instruments Components, Containers Packaging, Electrical Equipment, Media, Commercial Services Supplies, Food Staples Retailing, Professional Services, Insurance, Auto Components, Hotels Restaurants Leisure, Pharmaceuticals

Stocks - Tickers

CAH, PFG, CMCSA, ROP, SBNY, AVT, MTB, GWR, AGN, WEC, WU, KR, BLK, AVY, DE, NUE, HOG, MRK, TYL, WST, RHT, ADI, CMI, EIX, XEC, MAS, XEL, EBAY, EXPD, RGLD, EMR, BAC, RTX, STI, DISCK, PVH, OKE, AMP, ARG, BLL, PH, ODP, TMO, NVDA, D, C, RRC, K, ADP, WRB, WCN, AAP, EWBC, HIG, CTAS, UNFI, SEIC, STZ, SPLS, LRCX, L, VZ, THS, NCR, TFC, INTC, TNL, LBTYK, TROW, AMG, MSCI, KEY, DISCA, MAT, SBUX, ALL, WYNN, IBM, ORLY, EV, ALB, XRAY, PCP, TSCO, CL, CPRT, LVLT, CMA, OII, CA, AMZN, PG, CNC, UNM, NFX, SIVB, CHRW, HAS, FOSL, GNTX, BBY, BMY, SAM, ZBRA, ACM, SCHW, KO, FLS, EOG, WAB, UNP, DDD, GL, M, ROK, CPB, RL, WDC, BRCM, SNA, MDLZ, NLOK, TXN, AEE, SPGI, ADBE, DFS,

DXC, TXT, NI, MU, RS, APH, TRMB, OMC, LLTC, HES, CMS, ALGN, ED, QRTEA, SIRI, AIG, STLD, HUBB, ENDP, PRU, NSC, FE, HRL, LNT, TIF, RF, WEX, HFC, TPR, PFE, BR, BK, HWM, DGX, HUM, ARRS, DVA, DO, AME, WFC, FTI, FISV, DAL, PEP, A, BKNG, RTN, ITW, ASH, THC, IT, AON, DLTR, EMN, FCX, CSCO, NEU, J, COO, DKS, ZBH, PPL, COL, PNR, ALK, USB, FDS, TRV, TGT, PKI, KLAC, PNW, EVRG, HON, MS, NDSN, URI, CLX, PWR, QCOM, FIS, PKG, GGG, LHX, TEL, ABT, VVC, WM, JNPR, LUV, ILMN, AJG, GPS, GT, HAIN, RMD, FLIR, TSN, APD, HD, AVP, PDCO, ARW, ES, CSX, EW, BAX, FMC, ADSK, CB, ANDV, SWKS, PXD, MJN, PBCT, FAST, HPQ, ZION, NTAP, LOW, BSX, LM, IVZ, KMX, FITB, EXC, AFL, MCK, DCI, GD, RGA, R, SHW, RAI, ATI, VFC, IFF, WSM, MKC, CINF, VRSK, BA, HAL, COST, EPC, TDC, JNJ, EL, T, CAM, JOY, PTC, TSS, MAR, JCI, LMT, MET, NOV, TFX, BBBY, MCD, STJ, SNDK, EXPE, BEN, AAPL, LII, ANTM, EFX, AMAT, AZO, HP, ECL, BDX, CDNS, SIG, CVX, AYI, GS, SYY, XLNX, CFR, ACN, AABA, MRO, MMM, BWA, GIS, F, TAP, PCAR, PEG, IDXX, AVGO, GPN, PRGO, GE, CAG, KDP, PGR, CNP, JBHT, Y, AOS, TE, YUM, FLR, SRCL, ABC, IP, GME, NOC, DUK, TT, XRX, COF, PM, GNW, TER, TGNA, NEM, AKAM, CME, ROL, WBA, JBL, ADM, VRSN, VAR, LECO, MD, PII, HSY, MTD, ATR, HBI, LLY, SEE, NBL, KMB, FSLR, NDAQ, SRE, QRVO, LBTYA, X, HOLX, MDT, DHR, IPG, LH, VIAC, NKE, ICE, OI, APA, UNH, MAN, MUR, NTRS, VLO, AIZ, URBN, IPGP, MSFT, MSI, CTSH, HBAN, COP, SLM, UHS, LUMN, SCG, SO, ETFC, KSU, LNC, OXY, UAA, AEP, RPM, AN, CTXS, GWW, ADS, MCO, RHI, PNC, RJF, NEE, BCR, JWN, CI, RIG, JPM, FTR, FTNT, MMC, PPG, XOM, STT, ESRX, SWN, BRO, MA, DRI, WMB, PAYX, SJM, DVN, INTU, DD, CVS, KSS, CRM, DISH, CRI, MO, INGR, CCL, SWK, STX, APC, V, TJX, RCL, RE, ORCL, CAT, EQT, ETN, JBLU, FDX, MXIM, ANSS, GLW, AXP, PCG, DPZ, POM, UPS, ODFL, BC, MCHP, FFIV, PBI, WAT, TDY, EGN, JKHY, DG, SYK, MOS, DTE, AGCO, HOT, SNPS, SLB, MNST, ULTI, DOV, CYH, CMG, WMT, RSG, ISRG, EMC, IEX, CSL, CHTR, CHD, BKR, ROST, ULTA, HSIC, CF, OGE, CNX, WFM, ETR, IBKR, FL

B.4.2 2016

Sectors - S&P 500 Sectorial Indices

Health Care, Utilities, Industrials, Consumer Discretionary, Financials, Information Technology, Consumer Staples, Materials, Communication Services, Energy

Industries - S&P 500 Industries Indices

Multiline Retail, Consumer Finance, Health Care Equipment Supplies, Internet Direct Marketing Retail, Chemicals, Machinery, Textiles Apparel Luxury Goods, Household Products, Health Care Providers Services, Air Freight Logistics, Capital Markets, Software, Banks, Metals Mining, Building

Products, Trading Companies Distributors, Technology Hardware Storage Peripherals, Multi-Utilities, Industrial Conglomerates, Specialty Retail, Life Sciences Tools Services, Construction Engineering, Road Rail, Automobiles, Aerospace Defense, Energy Equipment Services, Electric Utilities, Leisure Products, Semiconductors Semiconductor Equipment, Beverages, Airlines, Communications Equipment, Food Products, IT Services, Diversified Telecommunication Services, Tobacco, Oil Gas Consumable Fuels, Electronic Equipment Instruments Components, Containers Packaging, Electrical Equipment, Media, Commercial Services Supplies, Food Staples Retailing, Professional Services, Insurance, Auto Components, Hotels Restaurants Leisure, Pharmaceuticals

Stocks - Tickers

CAH, PFG, CMCSA, ROP, SBNY, AVT, MTB, AGN, BIO, WEC, WU, KR, BLK, AVY, DE, NUE, HOG, MRK, TYL, WST, RHT, ADI, CMI, EIX, XEC, MAS, XEL, WRK, EBAY, EXPD, EMR, BAC, CASY, RTX, FLO, STI, DISCK, PVH, OKE, AMP, BLL, PH, ODP, TMO, NVDA, D, C, RRC, K, ADP, WRB, WCN, AAP, EWBC, HIG, CTAS, SEIC, STZ, SPLS, LRCX, L, VZ, NCR, TFC, INTC, TNL, LBTYK, TROW, AMG, MSCI, KEY, DISCA, MAT, CBOE, SBUX, ALL, WYNN, IBM, ORLY, ALB, XRAY, TSCO, CL, CPRT, LVLT, CMA, CA, AMZN, PG, CNC, UNM, NFX, SIVB, CHRW, HAS, FOSL, GNTX, BBY, BMY, ZBRA, ACM, SCHW, KO, FLS, EOG, WAB, UNP, GL, M, ROK, CPB, RL, WDC, SNA, MDLZ, NLOK, TXN, AEE, SPGI, ADBE, DFS, DXC, TXT, NI, MU, RS, APH, TRMB, GM, OMC, LLTC, HES, CMS, ALGN, OZK, ED, QRTEA, SIRI, AIG, STLD, HUBB, ENDP, PRU, NSC, FE, HRL, LNT, TIF, RF, HFC, TPR, PFE, BR, BK, HWM, DGX, HUM, ARRS, DVA, DO, AME, WFC, FTI, FISV, DAL, PEP, PACW, A, BKNG, RTN, ITW, THC, IT, AON, DLTR, EMN, FCX, CSCO, NEU, J, COO, MANH, ZBH, PPL, COL, PNR, ALK, USB, TTC, FDS, TRV, TGT, PKI, KLAC, CRL, PNW, EVRG, HON, MS, NDSN, URI, CLX, PWR, QCOM, FIS, PKG, GGG, LHX, TEL, ABT, CBSH, WM, JNPR, LUV, ILMN, AJG, GPS, GT, HAIN, RMD, FLIR, TSN, APD, HD, PDCO, ARW, ES, CSX, EW, BAX, FMC, ADSK, CB, ANDV, SWKS, PXD, MJN, PBCT, FAST, HPQ, ZION, NTAP, LOW, BSX, LM, IVZ, KMX, FITB, EXC, AFL, MCK, GD, RGA, R, SHW, MKTX, ZD, RAI, VFC, IFF, WSM, MKC, CINF, VRSK, BA, HAL, COST, TDC, JNJ, T, PTC, TSS, MAR, JCI, LMT, MET, SNV, NOV, TFX, BBBY, MCD, STJ, EXPE, BEN, AAPL, LII, ANTM, EFX, AMAT, AZO, HP, ECL, BDX, CDNS, SIG, CVX, AYI, GS, SYY, XLNX, ACN, AABA, MRO, MMM, BWA, GIS, F, TAP, PCAR, PEG, IDXX, AVGO, GPN, PRGO, GE, CAG, KDP, PGR, CNP, JBHT, Y, AOS, YUM, FLR, SRCL, ABC, IP, GME, NOC, DUK, TT, XRX, COF, PM, TER, TGNA, NEM, AKAM, CME, ROL, WBA, JBL, ADM, VRSN, VAR, MD, PII, HSY, MTD, ATR, HBI, LLY, IDTI, SEE, NBL, KMB, FSLR, NDAQ, SRE, QRVO, LBTYA, HOLX, MDT, DHR, IPG, LH, VIAC, NKE, ICE, OI, APA, UNH, MAN, MUR, NTRS, VLO, AIZ, URBN, IPGP, MSFT, MSI, CTSH, HBAN, COP, UHS,

LUMN, SCG, SO, ETFC, KSU, RNR, LNC, OXY, UAA, AEP, RPM, AN, CTXS, GWW, ADS, MCO, RHI, PNC, RJF, LYB, NEE, BCR, JWN, CI, RIG, JPM, FTR, FTNT, MMC, PPG, XOM, STT, ESRX, STE, SWN, BRO, MA, DRI, UAL, WMB, PAYX, SJM, DVN, INTU, DD, CVS, KSS, CRM, DISH, CRI, MO, INGR, CCL, SWK, STX, APC, V, TJX, RCL, ABMD, RE, ORCL, CAT, EQT, ETN, JBLU, FDX, MXIM, ANSS, GLW, AXP, PCG, DPZ, NLSN, AFG, ODFL, UPS, BC, MCHP, FFIV, PBI, WAT, JKHY, DG, SYK, MOS, DTE, SNPS, SLB, MNST, ULTI, DOV, SKX, CMG, WMT, RSG, ISRG, IEX, CSL, CHTR, CHD, TSLA, BKR, ROST, ULTA, NWS, HSIC, CF, OGE, CNX, WFM, ETR, IBKR, FL

B.4.3 2017

Sectors - S&P 500 Sectorial Indices

Health Care, Utilities, Industrials, Consumer Discretionary, Financials, Information Technology, Consumer Staples, Materials, Communication Services, Energy

Industries - S&P 500 Industries US Indices

Multiline Retail, Consumer Finance, Health Care Equipment Supplies, Internet Direct Marketing Retail, Chemicals, Machinery, Textiles Apparel Luxury Goods, Household Products, Health Care Providers Services, Air Freight Logistics, Capital Markets, Software, Banks, Metals Mining, Building Products, Trading Companies Distributors, Technology Hardware Storage Peripherals, Multi-Utilities, Industrial Conglomerates, Specialty Retail, Life Sciences Tools Services, Construction Engineering, Road Rail, Automobiles, Aerospace Defense, Energy Equipment Services, Electric Utilities, Leisure Products, Semiconductors Semiconductor Equipment, Beverages, Airlines, Communications Equipment, Food Products, IT Services, Diversified Telecommunication Services, Tobacco, Oil Gas Consumable Fuels, Electronic Equipment Instruments Components, Containers Packaging, Electrical Equipment, Media, Commercial Services Supplies, Food Staples Retailing, Professional Services, Insurance, Auto Components, Hotels Restaurants Leisure, Pharmaceuticals

Stocks - Tickers

CAH, PFG, CMCSA, ROP, SBNY, AVT, MTB, AGN, BIO, WEC, WU, KR, BLK, AVY, DE, NUE, HOG, MRK, TYL, CGNX, WST, RHT, ADI, CMI, EIX, XEC, MAS, XEL, WRK, EBAY, EXPD, EMR, BAC, RTX, STI, DISCK, PVH, OKE, AMP, BLL, PH, TMO, NVDA, D, C, RRC, K, ADP, WRB, AAP, EWBC, HIG, CTAS, SEIC, STZ, LRCX, L, VZ, NCR, TFC, INTC, TNL, LBTYK, TROW, AMG, MSCI, KEY, MDU, DISCA, MAT, CBOE, SBUX, ALL, WYNN, IBM, ORLY, ALB, XRAY, TSCO, CL, CPRT, CMA, CA, AMZN, PG, CNC, UNM, NFX, SIVB, CHRW, HAS, GNTX, BBY, BMY, ACM, SCHW, KO, FLS, EOG, WAB, UNP, GL, M, ROK, CPB, RL, WDC, SNA, MDLZ, NLOK, TXN, AEE,

SPGI, ADBE, DFS, DXC, TXT, NI, MU, RS, APH, TRMB, GM, OMC, HES, CMS, ALGN, OZK, ED, QRTEA, SIRI, AIG, STLD, HUBB, ENDP, PRU, NSC, FE, HRL, LNT, TIF, RF, WEX, HFC, TPR, PFE, BR, BK, HWM, DGX, HUM, ARRS, DVA, AME, WFC, FTI, FISV, DAL, PEP, PACW, A, BKNG, RTN, ITW, IT, AON, DLTR, EMN, FCX, FBHS, CSCO, KMI, J, COO, DKS, ZBH, PPL, COL, PNR, ALK, USB, TTC, XYL, FDS, TRV, TGT, PKI, KLAC, PNW, EVRG, HON, MS, NDSN, URI, CLX, PWR, QCOM, FIS, PKG, LHX, TEL, ABT, CBSH, WM, JNPR, LUV, ILMN, AJG, GPS, GT, RMD, FLIR, TSN, APD, HD, PDCO, ARW, ES, CSX, EW, BAX, FMC, ADSK, CB, ANDV, SWKS, PXD, PBCT, THO, FAST, HPQ, ZION, NTAP, LOW, BSX, IVZ, KMX, FITB, EXC, AFL, MCK, DCI, GD, RGA, R, SHW, MKTX, VFC, IFF, MKC, CINF, VRSK, BA, HAL, COST, TDC, JNJ, T, SMG, PTC, TSS, MAR, JCI, LMT, MET, NOV, TFX, BBBY, MCD, EXPE, BEN, CXO, AAPL, LII, ANTM, EFX, AMAT, AZO, HP, ECL, BDX, CDNS, SIG, CVX, AYI, GS, SYY, XLNX, CFR, ACN, AABA, MRO, MMM, BWA, GIS, F, TAP, PCAR, PEG, IDXX, HII, AVGO, GPN, PRGO, GE, AMD, CAG, KDP, PGR, CNP, JBHT, Y, AOS, YUM, FLR, SRCL, ABC, IP, NOC, DUK, TT, XRX, COF, PM, TER, TGNA, NEM, AKAM, CME, ROL, WBA, HCA, ADM, VRSN, VAR, LECO, MD, PII, HSY, MTD, HBI, LLY, SEE, NBL, KMB, FSLR, NDAQ, SRE, TDG, QRVO, LBTYA, X, HOLX, MDT, DHR, IPG, LH, VIAC, NKE, ICE, APA, UNH, MAN, MUR, NTRS, VLO, AIZ, LDOS, URBN, IPGP, MSFT, MSI, CTSH, HBAN, COP, SLM, UHS, LUMN, SCG, SO, ETFC, KSU, RNR, LNC, OXY, UAA, AEP, RPM, AN, CTXS, GWW, ADS, MCO, RHI, PNC, RJF, LYB, NEE, JWN, CI, RIG, JPM, FTR, FTNT, MMC, PPG, XOM, STT, ESRX, STE, SWN, BRO, MA, DRI, UAL, WMB, PAYX, SJM, DVN, INTU, DD, CVS, KSS, CRM, DISH, MO, INGR, CCL, SWK, STX, APC, V, TJX, RCL, ABMD, RE, ORCL, CAT, EQT, ETN, JBLU, FDX, APTV, MXIM, MPC, ANSS, GLW, AXP, PCG, DPZ, NLSN, AFG, ODFL, UPS, BC, MCHP, FFIV, PBI, WAT, CPRI, EGN, JKHY, DG, SYK, MOS, DTE, SNPS, SLB, MNST, ULTI, DOV, CMG, WMT, RSG, ISRG, IEX, CSL, CHTR, CHD, TSLA, BKR, ROST, ULTA, NWS, WCG, HSIC, CF, OGE, ETR, IBKR, FL

B.4.4 2018

Sectors - S&P 500 Sectorial Indices

Health Care, Utilities, Industrials, Consumer Discretionary, Financials, Information Technology, Consumer Staples, Materials, Communication Services, Energy

Industries - S&P 500 Industries US Indices

Multiline Retail, Consumer Finance, Health Care Equipment Supplies, Internet Direct Marketing Retail, Personal Products, Chemicals, Machinery, Textiles Apparel Luxury Goods, Household Products, Health Care Providers Services, Air Freight Logistics, Capital Markets, Software, Banks, Metals Mi-

ning, Building Products, Trading Companies Distributors, Technology Hardware Storage Peripherals, Multi-Utilities, Industrial Conglomerates, Specialty Retail, Life Sciences Tools Services, Construction Engineering, Road Rail, Automobiles, Aerospace Defense, Energy Equipment Services, Electric Utilities, Leisure Products, Semiconductors Semiconductor Equipment, Beverages, Airlines, Communications Equipment, Food Products, IT Services, Diversified Telecommunication Services, Oil Gas Consumable Fuels, Electronic Equipment Instruments Components, Containers Packaging, Electrical Equipment, Media, Commercial Services Supplies, Food Staples Retailing, Professional Services, Insurance, Auto Components, Hotels Restaurants Leisure, Pharmaceuticals

Stocks - Tickers

KNX, CAH, FICO, PFG, CMCSA, ROP, AVT, MTB, GWR, AGN, BIO, WEC, WU, KR, BLK, AVY, CY, DE, NUE, HOG, MRK, CGNX, RHT, ADI, CMI, EIX, XEC, MAS, XEL, WRK, EBAY, EXPD, EMR, BAC, CASY, RTX, STI, DISCK, PVH, OKE, AMP, MOH, BLL, PH, ICUI, TMO, NVDA, D, C, RRC, K, ADP, AAP, EWBC, HIG, CTAS, STZ, LRCX, L, VZ, NCR, TFC, INTC, TNL, LBTYK, TROW, AMG, MSCI, KEY, MDU, DISCA, MAT, CBOE, SBUX, ALL, WYNN, IBM, ORLY, EV, ALB, PSX, XRAY, TSCO, CL, CPRT, CMA, AMZN, NKTR, PG, CNC, UNM, NFX, CHRW, HAS, GNTX, BBY, BMY, ACM, SCHW, CMD, KO, FLS, EOG, UNP, GL, M, ROK, CPB, RL, WDC, SNA, MDLZ, NLOK, TXN, AEE, MELI, SPGI, ADBE, DFS, DXC, TXT, NI, MU, APH, GM, OMC, DAN, HES, CMS, ALGN, ED, QRTEA, SIRI, AIG, HUBB, PRU, NSC, FE, HRL, LNT, TIF, RF, HFC, TPR, EME, PFE, BR, BK, HWM, DGX, HUM, ARRS, DVA, AME, WFC, FTI, NCLH, MSM, FISV, DAL, PEP, A, BKNG, RTN, ITW, ASH, ITT, IT, AON, DLTR, EMN, FCX, FBHS, CSCO, NEU, KMI, J, COO, PPL, ZBH, PNR, ALK, USB, XYL, FDS, TRV, TGT, PKI, KLAC, CRL, PNW, IDA, EVRG, HON, MS, NDSN, URI, CLX, PWR, BLKB, FIS, QCOM, PKG, GGG, LHX, TEL, ABT, CBSH, WM, JNPR, LUV, ILMN, AJG, GPS, GT, HAIN, RMD, FLIR, TSN, APD, HD, PDCO, ARW, ES, CSX, EW, BAX, FMC, ADSK, CB, SWKS, PXD, PBCT, HRC, FAST, HPQ, ZION, NTAP, LOW, BSX, IVZ, KMX, FITB, EXC, AFL, MCK, DCI, GD, MKSI, MKTX, SHW, VFC, IFF, MKC, CINF, VRSK, BA, HAL, COST, FHN, JNJ, MPWR, T, TSS, MAR, JCI, KHC, LMT, MET, NOV, FAF, MCD, EXPE, BEN, CXO, AAPL, LII, ANTM, EFX, AMAT, AZO, HP, ECL, LFUS, BDX, CDNS, COHR, SIG, CVX, AYI, GS, SYY, XLNX, CFR, ACN, MRO, MMM, BWA, GIS, F, TAP, PCAR, PEG, IDXX, HII, AVGO, GPN, PRGO, GE, AMD, CAG, KDP, PGR, CNP, JBHT, AOS, WDAY, YUM, FLR, SRCL, ABC, IP, CR, NOC, DUK, TT, XRX, COF, NEM, MASI, AKAM, CME, WBA, HCA, JBL, ADM, VRSN, VAR, LECO, MD, HSY, MTD, ATR, HBI, LLY, SEE, NBL, KMB, FSLR, NDAQ, SRE, TDG, QRVO, LBTYA, CW, HOLX, MDT, DHR, IPG, LH, VIAC, NKE, ICE, APA, UNH, MAN, MUR, NTRS, VLO, AIZ, LDOS, IPGP, MSFT, MSI, CTSH, HBAN, COP, MMS, UHS, LUMN, SCG, SO, ETFC, KSU, LNC, OXY, UAA,

AEP, AN, CTXS, GWW, ADS, MCO, RHI, PNC, RJF, LYB, NEE, JWN, CI, JPM, FTNT, MMC, PPG, XOM, STT, BRO, MA, DRI, UAL, WMB, MGM, PAYX, SJM, DVN, INTU, DD, CVS, NATI, KSS, CRM, DISH, CRI, CLB, INGR, CCL, SWK, STX, APC, V, TJX, RCL, ABMD, RE, ORCL, CAT, EQT, ETN, JBLU, FDX, APTV, MXIM, MPC, ANSS, GLW, AXP, PCG, DPZ, NLSN, AFG, UPS, LOGM, BC, MCHP, FFIV, WAT, DNKN, CPRI, JKHY, DG, SYK, MOS, DTE, AGCO, SNPS, SLB, MNST, DOV, CMG, WMT, RSG, ISRG, IEX, CSL, CHTR, CHD, TSLA, BKR, EHC, ROST, ULTA, NWS, HSIC, CF, ETR, IBKR, FL

B.4.5 2019

Sectors - S&P 500 Sectorial Indices

Health Care, Utilities, Industrials, Consumer Discretionary, Financials, Information Technology, Consumer Staples, Materials, Communication Services, Energy

Industries - S&P 500 Industries US Indices

Multiline Retail, Consumer Finance, Health Care Equipment Supplies, Internet Direct Marketing Retail, Personal Products, Chemicals, Machinery, Textiles Apparel Luxury Goods, Household Products, Health Care Providers Services, Air Freight Logistics, Capital Markets, Software, Banks, Metals Mining, Building Products, Trading Companies Distributors, Technology Hardware Storage Peripherals, Multi-Utilities, Industrial Conglomerates, Specialty Retail, Life Sciences Tools Services, Construction Engineering, Road Rail, Automobiles, Aerospace Defense, Energy Equipment Services, Electric Utilities, Semiconductors Semiconductor Equipment, Beverages, Airlines, Communications Equipment, Food Products, IT Services, Diversified Telecommunication Services, Oil Gas Consumable Fuels, Electronic Equipment Instruments Components, Containers Packaging, Electrical Equipment, Media, Commercial Services Supplies, Food Staples Retailing, Professional Services, Insurance, Auto Components, Hotels Restaurants Leisure, Pharmaceuticals

Stocks - Tickers

KNX, CAH, FICO, PFG, CMCSA, CHE, ROP, MTB, AGN, BIO, WEC, WU, KR, BLK, AVY, DE, NUE, HOG, MRK, CGNX, ADI, CMI, EIX, XEC, MAS, XEL, WRK, EBAY, EXPD, EMR, BAC, CASY, RTX, DISCK, PVH, OKE, AMP, MOH, BLL, PH, ICUI, TMO, NVDA, D, C, K, ADP, AAP, EWBC, HIG, CTAS, STZ, LRCX, L, VZ, TFC, INTC, LBTYK, TROW, AMG, MSCI, KEY, MDU, DISCA, CBOE, SBUX, ALL, WYNN, IBM, ORLY, ALB, PSX, XRAY, TSCO, CL, CPRT, CMA, AMZN, NKTR, ALLE, PG, CNC, UNM, SIVB, CHRW, GNTX, BBY, BMY, ACM, SCHW, KO, FLS, EOG, UNP, GL, M, ROK, CPB, RL, WDC, SNA, MDLZ, NLOK, TXN, AEE, MELI, SPGI, ADBE,

DFS, DXC, TXT, NI, MU, APH, GM, OMC, HES, CMS, ALGN, ED, SIRI, AIG, HUBB, PRU, NSC, HLT, FE, HRL, LNT, TIF, RF, HFC, TPR, PFE, BR, BK, HWM, DGX, HUM, DVA, AME, WFC, FTI, NCLH, FISV, DAL, PEP, A, BKNG, RTN, ITW, ASH, ITT, IT, AON, DLTR, EMN, FCX, FBHS, GDOT, CSCO, NEU, KMI, J, COO, PPL, ZBH, PNR, ALK, USB, XYL, FDS, TRV, TGT, IQV, PKI, KLAC, CRL, PNW, IDA, EVRG, HON, MS, FLT, NDSN, URI, CLX, PWR, QCOM, FIS, PKG, GGG, LHX, TEL, ABT, FANG, CBSH, WM, JNPR, LUV, ILMN, AJG, GPS, GT, HAE, RMD, FLIR, TSN, APD, HD, ARW, ES, CSX, EW, BAX, FMC, ADSK, CB, SWKS, PXD, PBCT, HRC, FAST, HPQ, ZION, FRC, NTAP, LOW, BSX, CIEN, IVZ, KMX, FITB, EXC, AFL, MCK, DCI, GD, SHW, MKTX, VFC, IFF, MKC, CINF, VRSK, BA, HAL, COST, JNJ, MPWR, T, MAR, JCI, KHC, LMT, MET, NOV, FAF, LANC, MCD, EXPE, BEN, CXO, AAPL, LII, ANTM, EFX, AMAT, AZO, HP, ECL, LFUS, BDX, CDNS, CVX, AYI, GS, SYY, XLNX, CFR, ACN, MRO, MMM, BWA, GIS, F, TAP, PCAR, PEG, IDXX, HII, AVGO, GMED, GPN, PRGO, GE, NWSA, AMD, CAG, KDP, PGR, ZTS, JBHT, CNP, AOS, WDAY, YUM, FLR, ABC, IP, CR, NOC, DUK, TT, XRX, COF, LIN, NEM, MASI, AKAM, CME, ROL, WBA, HCA, ADM, VRSN, VAR, LECO, CE, HSY, MTD, ATR, HBI, LLY, SEE, NBL, KMB, FSLR, NDAQ, SRE, TDG, QRVO, LBTYA, CW, HOLX, MDT, DHR, IPG, LH, VIAC, LULU, NKE, ICE, APA, UNH, NTRS, VLO, AIZ, LDOS, IPGP, MSFT, MSI, CTSH, HBAN, COP, UHS, LUMN, SO, ETFC, KSU, LNC, OXY, UAA, AEP, CTXS, GWW, ADS, MCO, RHI, PNC, RJF, LYB, NEE, JWN, CI, JPM, FTNT, AAL, MMC, PPG, XOM, STT, BRO, MA, DRI, UAL, WMB, MGM, PAYX, SJM, DVN, INTU, DD, CVS, NATI, KSS, CRM, DISH, INGR, CCL, SWK, STX, V, TJX, RCL, ABMD, RE, ORCL, CAT, EQT, ETN, JBLU, FDX, APTV, MXIM, MPC, ANSS, GLW, AXP, PCG, DPZ, NLSN, AFG, UPS, LOGM, MCHP, FFIV, WAT, DNKN, CPRI, JKHY, DG, SYK, MOS, DTE, AGCO, SNPS, SLB, MNST, DOV, FIVE, CMG, WMT, RSG, ISRG, IEX, CSL, CHTR, CHD, TSLA, BKR, EHC, ROST, ULTA, NWS, WCG, HSIC, CF, ETR, IBKR, FL

B.4.6 2020

Sectors - S&P 500 Sectorial Indices

Health Care, Utilities, Industrials, Consumer Discretionary, Financials, Information Technology, Consumer Staples, Materials, Communication Services, Energy

Industries - SP Industrials US Indices

Multiline Retail, Consumer Finance, Health Care Equipment Supplies, Internet Direct Marketing Retail, Personal Products, Chemicals, Machinery, Textiles Apparel Luxury Goods, Household Products, Health Care Providers Services, Air Freight Logistics, Capital Markets, Software, Banks, Metals Mining, Building Products, Trading Companies Distributors, Technology Hardware Storage Peripherals,

Multi-Utilities, Industrial Conglomerates, Specialty Retail, Life Sciences Tools Services, Construction Engineering, Road Rail, Automobiles, Aerospace Defense, Energy Equipment Services, Electric Utilities, Semiconductors Semiconductor Equipment, Beverages, Airlines, Communications Equipment, Food Products, IT Services, Diversified Telecommunication Services, Oil Gas Consumable Fuels, Electronic Equipment Instruments Components, Containers Packaging, Electrical Equipment, Media, Commercial Services Supplies, Food Staples Retailing, Professional Services, Insurance, Auto Components, Hotels Restaurants Leisure, Pharmaceuticals

Stocks - Tickers

KNX, CAH, FICO, PFG, CMCSA, CHE, ROP, MTB, INFO, BIO, WEC, CSGP, KR, BLK, WU, AVY, DE, NUE, HOG, MRK, CGNX, ADI, CMI, AMCR, EIX, XEC, MAS, XEL, WRK, EBAY, EXPD, EMR, BAC, CASY, RTX, DISCK, PVH, OKE, AMP, MOH, BLL, PH, TMO, NVDA, D, C, K, ADP, WRB, AAP, ANET, EWBC, HIG, CTAS, CTLT, STZ, LRCX, L, VZ, TFC, INTC, LBTYK, CDK, TROW, MSCI, KEY, MDU, DISCA, CBOE, SBUX, ALL, WYNN, IBM, LVS, ORLY, ALB, PSX, XRAY, TSCO, CL, CPRT, CMA, AMZN, ALLE, PG, CNC, UNM, SIVB, CHRW, GNTX, BBY, BMY, ZBRA, ACM, SCHW, KO, FLS, EOG, WAB, UNP, GL, CACI, M, ROK, CPB, RL, WDC, SNA, MDLZ, NLOK, TXN, AEE, MELI, SPGI, ADBE, DFS, DXC, TXT, NI, MU, APH, GM, OMC, HES, CMS, ALGN, ED, SIRI, AIG, HUBB, PRU, NSC, HLT, FE, HRL, LNT, TIF, RF, HFC, TPR, PFE, BR, BK, HWM, DGX, HUM, DVA, AME, WFC, CFG, FTI, NCLH, FISV, DAL, PEP, A, BKNG, ITW, ITT, IT, AON, DLTR, EMN, SPLK, FCX, FBHS, SYF, CSCO, KMI, J, COO, PPL, ZBH, PNR, ALK, USB, XYL, FDS, TRV, TGT, IQV, PKI, KLAC, CRL, PNW, EVRG, HON, MS, FLT, NDSN, URI, CLX, PWR, NOW, FIS, QCOM, PKG, GGG, LHX, TEL, ABT, FANG, CBSH, WM, JNPR, LUV, ILMN, AJG, GPS, HAE, RMD, FLIR, TSN, APD, HD, ARW, ES, CSX, EW, BAX, FMC, ADSK, CB, SWKS, PXD, PBCT, HRC, FAST, HPQ, ZION, FRC, NTAP, LOW, BSX, CIEN, IVZ, KMX, FITB, EXC, AFL, MCK, DCI, GD, MKSI, MKTX, SHW, VFC, IFF, MKC, CINF, VRSK, BA, HAL, COST, CDW, JNJ, MPWR, T, MAR, JCI, KHC, LMT, MET, NOV, FAF, TFX, MCD, EXPE, BEN, CXO, AAPL, LII, ANTM, EFX, AMAT, AZO, HP, ECL, BDX, CDNS, CVX, GS, SYY, XLNX, CFR, ACN, MRO, MMM, BWA, GIS, F, TAP, PCAR, PEG, IDXX, HII, AVGO, GMED, GPN, PRGO, GE, LEA, NWSA, AMD, CAG, KDP, PGR, ZTS, JBHT, CNP, AOS, WDAY, YUM, ABC, IP, NOC, DUK, TT, XRX, KEYS, COF, LIN, NEM, MASI, AKAM, CME, ROL, WBA, HCA, JBL, ADM, VRSN, VAR, LECO, CE, HSY, MTD, ATR, HBI, LLY, SEE, FSLR, KMB, NDAQ, SRE, TDG, QRVO, LBTYA, CW, HOLX, MDT, DHR, IPG, LH, VIAC, LULU, NKE, ICE, APA, UNH, NTRS, VLO, AIZ, LDOS, IPGP, MSFT, MSI, CTSH, COLM, HBAN, COP, UHS, LUMN, SO, KSU, LNC, OXY, UAA, AEP, CTXS, GWW, ADS, MCO, RHI, PNC, RJF, LYB, NEE, JWN, CI, JPM, FTNT, AAL, MMC, PPG, XOM, STT, STE,

BRO, MA, DRI, UAL, WMB, MGM, PAYX, SJM, DVN, INTU, DD, CVS, KSS, CRM, DISH, INGR, CCL, SWK, CHH, STX, V, TJX, RCL, ABMD, RE, ORCL, CAT, ETN, FDX, APTV, MXIM, MPC, ANSS, GLW, AXP, DPZ, NLSN, AFG, ODFL, UPS, MCHP, FFIV, WAT, CPRI, JKHY, DG, SYK, MOS, DTE, SNPS, SLB, MNST, DOV, FIVE, CMG, WMT, RSG, ISRG, IEX, CSL, CHTR, CHD, TSLA, BKR, EHC, ROST, ULTA, NWS, HSIC, CF, ETR, IBKR

B.4.7 2021

Sectors - S&P 500 Sectorial Indices

Health Care, Utilities, Industrials, Consumer Discretionary, Financials, Information Technology, Consumer Staples, Materials, Communication Services, Energy

Industries - S&P 500 Industries US Indices

Multiline Retail, Consumer Finance, Health Care Equipment Supplies, Internet Direct Marketing Retail, Personal Products, Chemicals, Machinery, Textiles Apparel Luxury Goods, Household Products, Health Care Providers Services, Air Freight Logistics, Capital Markets, Software, Banks, Metals Mining, Building Products, Trading Companies Distributors, Technology Hardware Storage Peripherals, Multi-Utilities, Industrial Conglomerates, Specialty Retail, Life Sciences Tools Services, Construction Engineering, Road Rail, Automobiles, Aerospace Defense, Energy Equipment Services, Electric Utilities, Semiconductors Semiconductor Equipment, Beverages, Airlines, Communications Equipment, Food Products, IT Services, Diversified Telecommunication Services, Oil Gas Consumable Fuels, Electronic Equipment Instruments Components, Containers Packaging, Electrical Equipment, Media, Commercial Services Supplies, Food Staples Retailing, Professional Services, Insurance, Auto Components, Hotels Restaurants Leisure, Pharmaceuticals

Stocks - Tickers

KNX, CAH, FICO, PFG, CMCSA, CHE, ROP, MTB, INFO, BIO, WU, KR, BLK, AVY, DE, NUE, MRK, TYL, CGNX, WST, ADI, AMCR, EIX, XEL, MAS, WRK, EBAY, EXPD, EMR, BAC, CASY, RTX, DISCK, PVH, OKE, AMP, MOH, BLL, TMO, NVDA, AXON, D, C, K, ADP, WRB, AAP, ANET, EWBC, HIG, CTAS, CTLT, STZ, LRCX, L, VZ, CDK, AMED, TROW, MSCI, KEY, DISCA, CBOE, SBUX, ALL, WYNN, IBM, LVS, ORLY, ALB, PSX, CABO, XRAY, TSCO, CL, CPRT, CMA, AMZN, ALLE, PG, CNC, UNM, LAD, SIVB, CHRW, GNTX, BBY, BMY, ZBRA, ACM, SCHW, KO, FLS, EOG, WAB, UNP, GL, CACI, CPB, RL, WDC, SNA, MDLZ, NLOK, TXN, AEE, MELI, SPGI, ADBE, DFS, DXC, TXT, NI, MU, APH, GM, OMC, HES, ALGN, ED, SIRI, AIG, PRU, HLT, FE, HRL, LNT, RF, HFC, TPR, ETSY, PFE, BR, BK, DGX, HUM, DVA, AME, WFC, CFG, FTI, NCLH, FISV, DAL,

PEP, A, BKNG, ITW, ITT, IT, AON, DLTR, EMN, SPLK, FCX, FBHS, SYF, CSCO, KMI, J, COO, MANH, ZBH, PPL, PNR, ALK, USB, XYL, FDS, TRV, TGT, IQV, PKI, CRL, PNW, EVRG, MS, FLT, URI, CLX, PWR, NOW, FIS, QCOM, PKG, LHX, TEL, ABT, FANG, CBSH, WM, JNPR, LUV, ILMN, AJG, GPS, RMD, TSN, APD, HD, ARW, CHDN, CSX, IIVI, EW, BAX, FMC, ADSK, CB, SWKS, PXD, PBCT, FAST, HPQ, ZION, FRC, JAZZ, NTAP, LOW, BSX, CIEN, IVZ, DXCM, KMX, FITB, EXC, AFL, MCK, GD, MKSI, MKTX, SHW, VFC, IFF, MKC, CINF, VRSK, BA, HAL, COST, FHN, CDW, PYPL, CLF, MPWR, T, CZR, MAR, JCI, KHC, LMT, MET, HPE, NOV, TFX, MCD, EXPE, BEN, AAPL, LII, ANTM, AMAT, AZO, ECL, LFUS, BDX, CDNS, CVX, GS, SYY, XLNX, ACN, MRO, MMM, BWA, GIS, F, TAP, PCAR, PEG, IDXX, HII, AVGO, GMED, GPN, PRGO, GE, LEA, NWSA, AMD, CAG, KDP, PGR, ZTS, JBHT, CNP, WDAY, ABC, GNRC, DECK, NOC, TT, XRX, KEYS, COF, MRVL, LIN, NEM, MASI, LHCG, AKAM, CME, ROL, HCA, JBL, ADM, VRSN, LECO, CE, HSY, MTD, ATR, HBI, LLY, SEE, FSLR, KMB, NDAQ, SRE, TDG, QRVO, HOLX, MDT, DHR, IPG, LH, VIAC, LULU, NKE, ICE, APA, UNH, NTRS, AIZ, LDOS, IPGP, MSFT, MSI, CTSH, HBAN, LUMN, LITE, LNC, OXY, UAA, AEP, CTXS, MCO, RHI, PNC, RJF, LYB, NEE, CI, JPM, FTNT, AAL, MMC, XOM, STT, STE, MA, DRI, UAL, WMB, MGM, PAYX, SJM, DVN, INTU, DD, CVS, CRM, DISH, MIDD, CCL, SWK, STX, V, TJX, RCL, ABMD, RE, ORCL, CAT, ETN, APTV, MPC, ANSS, GLW, DAR, AXP, PAYC, DPZ, NLSN, UPS, ODFL, MCHP, FFIV, ENPH, WAT, TDY, CPRI, JKHY, DG, SYK, MOS, DTE, AGCO, SNPS, SLB, MNST, FIVE, CMG, WMT, RSG, ISRG, IEX, CHTR, CHD, TSLA, BKR, EHC, ROST, ULTA, NWS, HSIC, CF, ETR, IBKR

Bibliography

Andersen, T., Bollerslev, T., Christoffersen, P., & Diebold, F. (2005). *Volatility forecasting* (Rapport technique). National Bureau of Economic Research, Inc.

Asness, C. S., Moskowitz, T. J., & Pedersen, L. H. (2013). Value and momentum everywhere. *The Journal of Finance*, *68*(3), 929–985.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization.

Bahdanau, D., Cho, K., & Bengio, Y. (2017). Neural machine translation by jointly learning to align and translate.

Bahdanau, D., Cho, K. H., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd international conference on learning representations, iclr 2015.*

Bakker, B., et al. (2001). Reinforcement learning with long short-term memory. In *Nips* (pp. 1475–1482).

Baldi, P., & Brunak, S. (2001). *Bioinformatics : the machine learning approach*. MIT press.

Barroso, P., & Santa-Clara, P. (2015). Momentum has its moments. *Journal of Financial Economics*, *116*(1), 111–120.

Basak, S., Kar, S., Saha, S., Khaidem, L., & Dey, S. R. (2019). Predicting the direction of stock market prices using tree-based classifiers. *The North American Journal of Economics and Finance*, *47*, 552–567.

Baum, L. E., & Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, *37*(6), 1554–1563.

Bayram, M., & Baraniuk, R. G. (1996). Multiple window time-frequency analysis. In *Proceedings of third international symposium on time-frequency and time-scale analysis (tfts-96)* (pp. 173–176).

Baz, J., Granger, N., Harvey, C. R., Le Roux, N., & Rattray, S. (2015). Dissecting investment strategies in the cross section and time series. *Available at SSRN 2695101*.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157–166.

Black, F., & Litterman, R. (1992). Global portfolio optimization. *Financial analysts journal*, *48*(5), 28–43.

Blitz, D., Huij, J., & Martens, M. (2011). Residual momentum. *Journal of Empirical Finance*, *18*(3), 506–521.

Britz, D., Goldie, A., Luong, M.-T., & Le, Q. (2017). Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 conference on empirical methods in natural language processing* (pp. 1442–1451).

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... others (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Bruch, S. (2021). An alternative cross entropy loss for learning-to-rank. In *Proceedings of the web conference 2021* (pp. 118–126).

Bruch, S., Wang, X., Bendersky, M., & Najork, M. (2019). An analysis of the softmax cross entropy loss for learning-to-rank with binary relevance. In *Proceedings of the 2019 acm sigir international conference on theory of information retrieval* (pp. 75–78).

Burges, C., Ragno, R., & Le, Q. (2006). Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems*, *19*, 193–200.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., & Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on machine learning* (pp. 89–96).

Cao, L., & Tay, F. E. (2001). Financial forecasting using support vector machines. *Neural Computing & Applications*, *10*(2), 184–192.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). Learning to rank : from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on machine learning* (pp. 129–136).

Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., & Sutskever, I. (2020). Generative pretraining from pixels. In *International conference on machine learning* (pp. 1691–1703).

Chichportich, J., Elie, R., Kharroubi, I., & De Servigny, A. (2020). Optimal quantized belief propagation. *Preprint*.

Cho, K., Courville, A., & Bengio, Y. (2015). Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, *11*(17), 1875–1886.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on empirical methods in natural language processing (emnlp 2014).*

Church, K. W. (2017). Word2vec. *Natural Language Engineering*, *23*(1), 155–162.

Clenow, A. F. (2015). *Stocks on the move : beating the market with hedge fund momentum strategies*. Smashwords Edition.

Cohen, L. (1989). Time-frequency distributions-a review. *Proceedings of the IEEE*, *77*(7), 941–981.

Daniel, K. (2017). *Thinking, fast and slow.*

Deco, G., & Rolls, E. T. (2005). Neurodynamics of biased competition and cooperation for attention : a model with spiking neurons. *Journal of neurophysiology*, *94*(1), 295–313.

Dehaene, S. (2012). Les grands principes de l'apprentissage. *Collège de France*, *20*.

Desai, A., Freeman, C., Wang, Z., & Beaver, I. (2021). Timevae : A variational auto-encoder for multivariate time series generation.

Durbin, R., Eddy, S. R., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis : probabilistic models of proteins and nucleic acids*. Cambridge university press.

Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, *4*(Nov), 933–969.

Freyberger, J., Neuhierl, A., & Weber, M. (2020). Dissecting characteristics nonparametrically. *The Review of Financial Studies*, *33*(5), 2326–2377.

Fuhr, N. (1989). Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems (TOIS)*, *7*(3), 183–204.

Gao, L., Han, Y., Li, S. Z., & Zhou, G. (2018). Market intraday momentum. *Journal of Financial Economics*, *129*(2), 394–414.

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2000). *Database system implementation* (Vol. 672). Prentice Hall Upper Saddle River.

Geczy, C. C., & Samonov, M. (2016). Two centuries of price-return momentum. *Financial Analysts Journal*, *72*(5), 32–56.

Georgopoulou, A., & Wang, J. (2017). The trend is your friend : Time-series momentum strategies across equity and commodity markets. *Review of Finance*, *21*(4), 1557–1592.

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget : Continual prediction with lstm. *Neural computation*, *12*(10), 2451–2471.

Gey, F. C. (1994). Inferring probability of relevance using the method of logistic regression. In *Sigir'94* (pp. 222–231).

Gou, J., Yu, B., Maybank, S. J., & Tao, D. (2021). Knowledge distillation : A survey. *International Journal of Computer Vision*, *129*(6), 1789–1819.

Graf, S., & Luschgy, H. (2000). *Foundations of quantization for probability distributions*. Springer.

Graves, A., Mohamed, A.-r., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 ieee international conference on acoustics, speech and signal processing* (pp. 6645–6649).

Gregor, K., Danihelka, I., Graves, A., Rezende, D., & Wierstra, D. (2015). Draw : A recurrent neural network for image generation. In *International conference on machine learning* (pp. 1462–1471).

Gupta, A., & Dhingra, B. (2012). Stock market prediction using hidden markov models. In *2012 students conference on engineering and systems* (pp. 1–4).

Herbrich, R., Graepel, T., Obermayer, K., et al. (2000). Large margin rank boundaries for ordinal regression. *Advances in large margin classifiers*, *88*(2), 115–132.

Herbrich, R., Minka, T., & Graepel, T. (2006). Trueskill™ : a bayesian skill rating system. *Advances in neural information processing systems*, *19*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Huang, W., Lai, K. K., Nakamori, Y., Wang, S., & Yu, L. (2007). Neural networks in finance and economics forecasting. *International Journal of Information Technology & Decision Making*, *6*(01), 113–140.

Ioffe, S., & Szegedy, C. (2015). Batch normalization : Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448–456).

Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers : Implications for stock market efficiency. *The Journal of finance*, *48*(1), 65–91.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the eighth acm sigkdd international conference on knowledge discovery and data mining* (pp. 133–142).

Kendall, M., & Gibbons, J. (1990). *Rank correlation methods, oxford university press.* London.

Kenton, J. D. M.-W. C., & Toutanova, L. K. (2019). Bert : Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacl-hlt* (pp. 4171–4186).

Kim, K.-j. (2003). Financial time series forecasting using support vector machines. *Neurocomputing*, *55*(1-2), 307–319.

Kim, S., Shephard, N., & Chib, S. (1998). Stochastic volatility : likelihood inference and comparison with arch models. *The review of economic studies*, *65*(3), 361–393.

Kingma, D. P., & Ba, J. (2015). Adam : A method for stochastic optimization. In *Iclr*.

Kingma, D. P., Mohamed, S., Jimenez Rezende, D., & Welling, M. (2014). Semi-supervised learning with deep generative models. *Advances in neural information processing systems*, *27*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv :1312.6114*.

Kombrink, S., Mikolov, T., Karafiát, M., & Burget, L. (2011). Recurrent neural network based language modeling in meeting recognition. In *Interspeech* (Vol. 11, pp. 2877–2880).

Krogh, A., Brown, M., Mian, I. S., Sjölander, K., & Haussler, D. (1994). Hidden markov models in computational biology : Applications to protein modeling. *Journal of molecular biology*, *235*(5), 1501–1531.

Kulkarni, T. D., Whitney, W. F., Kohli, P., & Tenenbaum, J. (2015). Deep convolutional inverse graphics network. *Advances in neural information processing systems*, *28*.

Ledoit, O., & Wolf, M. (2004). Honey, i shrunk the sample covariance matrix. *The Journal of Portfolio Management*, *30*(4), 110–119.

Li, P., Wu, Q., & Burges, C. (2007). Mcrank : Learning to rank using multiple classification and gradient boosting. *Advances in neural information processing systems*, *20*, 897–904.

Li, Y., & Cao, H. (2018). Prediction for tourism flow based on lstm neural network. *Procedia Computer Science*, *129*, 277–283.

Li, Y., Kaiser, L., Bengio, S., & Si, S. (2019). Area attention. In *International conference on machine learning* (pp. 3846–3855).

Lim, B. Y., Wang, J. G., & Yao, Y. (2018). Time-series momentum in nearly 100 years of stock returns. *Journal of Banking & Finance*, *97*, 283–296.

Lindsay, G. W. (2020). Attention in psychology, neuroscience, and machine learning. *Frontiers in computational neuroscience*, *14*, 29.

Liu, T.-Y., et al. (2009). Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, *3*(3), 225–331.

Longerstaey, J., & Spencer, M. (1996). Riskmetricstm—technical document. *Morgan Guaranty Trust Company of New York : New York*, *51*, 54.

Luong, M.-T., Pham, H., & Manning, C. D. (s. d.). Effective approaches to attention-based neural machine translation.

Mallat, S. (1999). *A wavelet tour of signal processing*. Elsevier.

Manning, C., & Schutze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

Manogaran, G., Vijayakumar, V., Varatharajan, R., Malarvizhi Kumar, P., Sundarasekar, R., & Hsu, C.-H. (2018). Machine learning based big data processing framework for cancer diagnosis using hidden markov model and gm clustering. *Wireless personal communications*, *102*(3), 2099–2116.

Markowitz, H. M., et al. (1952). (1952). portfolio selection. *Journal of Finance*, *7*(1), 77–91.

Martins, A., & Astudillo, R. (2016). From softmax to sparsemax : A sparse model of attention and multi-label classification. In *International conference on machine learning* (pp. 1614–1623).

Martins, A., Farinhas, A., Treviso, M., Niculae, V., Aguiar, P., & Figueiredo, M. (2020). Sparse and continuous attention mechanisms. *Advances in Neural Information Processing Systems*, *33*, 20989–21001.

McCann, B., Bradbury, J., Xiong, C., & Socher, R. (2017). Learned in translation : Contextualized word vectors. In *Advances in neural information processing systems* (pp. 6294–6305).

Menkhoff, L., Sarno, L., Schmeling, M., & Schrimpf, A. (2012). Currency momentum strategies. *Journal of Financial Economics*, *106*(3), 660–684.

Mezard, M., & Montanari, A. (2009). *Information, physics, and computation*. Oxford University Press.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing*

systems (pp. 3111–3119).

Minka, T. P. (2001). Expectation propagation for approximate bayesian inference. In *Proceedings of the 17th conference in uncertainty in artificial intelligence* (pp. 362–369).

Moskowitz, T. J., Ooi, Y. H., & Pedersen, L. H. (2012). Time series momentum. *Journal of financial economics*, *104*(2), 228–250.

Murthy, G., Allu, S. R., Andhavarapu, B., Bagadi, M., & Belusonti, M. (2020). Text based sentiment analysis using lstm. *Int. J. Eng. Res. Tech. Res*, *9*(05).

Nag, R., Wong, K., & Fallside, F. (1986). Script recognition using hidden markov models. In *Icassp'86. ieee international conference on acoustics, speech, and signal processing* (Vol. 11, pp. 2071–2074).

Nallapati, R. (2004). Discriminative models for information retrieval. In *Proceedings of the 27th annual international acm sigir conference on research and development in information retrieval* (pp. 64–71).

Nelson, D. M., Pereira, A. C., & De Oliveira, R. A. (2017). Stock market's price movement prediction with lstm neural networks. In *2017 international joint conference on neural networks (ijcnn)* (pp. 1419–1426).

Nti, I. K., Adekoya, A. F., & Weyori, B. A. (2020). A comprehensive evaluation of ensemble learning for stock-market prediction. *Journal of Big Data*, *7*(1), 1–40.

Nystrup, P., Madsen, H., & Lindström, E. (2017). Long memory of financial time series and hidden markov models with time-varying parameters. *Journal of Forecasting*, *36*(8), 989–1002.

Pagès, G. (2018). Numerical probability. In *Universitext.* Springer.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318).

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove : Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 1532–1543).

Poh, D., Lim, B., Zohren, S., & Roberts, S. (2021). Building cross-sectional systematic strategies by learning to rank. *The Journal of Financial Data Science*, *3*(2), 70–86.

Qin, T., Zhang, X.-D., Tsai, M.-F., Wang, D.-S., Liu, T.-Y., & Li, H. (2008). Query-level loss functions for information retrieval. *Information Processing & Management*, *44*(2), 838–855.

Rabiner, L., & Juang, B. (1986). An introduction to hidden markov models. *ieee assp magazine*, *3*(1), 4–16.

Radev, D. R., Qi, H., Wu, H., & Fan, W. (2002). Evaluating web-based question answering systems. In *Lrec.*

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, *21*, 1–67.

Reddi, S., Zaheer, M., Sachan, D., Kale, S., & Kumar, S. (2018). Adaptive methods for nonconvex optimization. In *Proceeding of 32nd conference on neural information processing systems (nips 2018).*

Reddi, S. J., Kale, S., & Kumar, S. (2018). On the convergence of adam and beyond. In *International conference on learning representations.*

Rezende, D. J., Mohamed, S., & Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning* (pp. 1278–1286).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Salimans, T., Kingma, D., & Welling, M. (2015). Markov chain monte carlo and variational inference : Bridging the gap. In *International conference on machine learning* (pp. 1218–1226).

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization ? *Advances in neural information processing systems*, *31*.

Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., ... others (2020). Improved protein structure prediction using potentials from deep learning. *Nature*, *577*(7792), 706–710.

Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning : A systematic literature review : 2005–2019. *Applied soft computing*, *90*, 106181.

Sohn, K., Lee, H., & Yan, X. (2015). Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, *28*.

Song, Q., Liu, A., & Yang, S. Y. (2017). Stock portfolio selection using learning-to-rank algorithms with news sentiment. *Neurocomputing*, *264*, 20–28.

Sood, E., Tannert, S., Müller, P., & Bulling, A. (2020). Improving natural language processing tasks with human gaze-guided neural attention. *Advances in Neural Information Processing Systems*, *33*, 6327–6341.

Sundermeyer, M., Schlüter, R., & Ney, H. (2012). Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association.*

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Swezey, R., Grover, A., Charron, B., & Ermon, S. (2021). Pirank : Scalable learning to rank via differentiable sorting. *Advances in Neural Information Processing Systems*, *34*.

Tang, G., Müller, M., Rios, A., & Sennrich, R. (2018). Why self-attention ? a targeted evaluation of neural machine translation architectures. In *2018 conference on empirical methods in natural language processing*.

Testa, A. C., Hane, J. K., Ellwood, S. R., & Oliver, R. P. (2015). Codingquarry : highly accurate hidden markov model gene prediction in fungal genomes using rna-seq transcripts. *BMC genomics*, *16*(1), 1–12.

Valizadegan, H., Jin, R., Zhang, R., & Mao, J. (2009). Learning to rank by optimizing ndcg measure. *Advances in neural information processing systems*, *22*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Veselỳ, K., Karafiát, M., Grézl, F., Janda, M., & Egorova, E. (2012). The language-independent bottleneck features. In *2012 ieee spoken language technology workshop (slt)* (pp. 336–341).

Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and tell : A neural image caption generator. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3156–3164).

Walker, J., Doersch, C., Gupta, A., & Hebert, M. (2016). An uncertain future : Forecasting from static images using variational autoencoders. In *European conference on computer vision* (pp. 835–851).

Wang, L. (2018). *Stock ranking with market microstructure, news and technical indicators* (Thèse de doctorat non publiée). University of Georgia.

Wang, P., Qian, Y., Soong, F. K., He, L., & Zhao, H. (2015). Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *arXiv preprint arXiv :1510.06168*.

Wang, Y., Wang, L., Li, Y., He, D., & Liu, T.-Y. (2013). A theoretical analysis of ndcg type ranking measures. In *Conference on learning theory* (pp. 25–54).

Wang, Z., Xiao, D., Fang, F., Govindan, R., Pain, C. C., & Guo, Y. (2018). Model identification of reduced order fluid dynamics systems using deep learning. *International Journal for Numerical Methods in Fluids*, *86*(4), 255–268.

Werbos, P. J. (1990). Backpropagation through time : what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, *1*(2), 270–280.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... Bengio, Y. (2015). Show, attend and tell : Neural image caption generation with visual attention. In *International conference on machine learning* (pp. 2048–2057).

Yuan, L., Chen, Y., Wang, T., Yu, W., Shi, Y., Jiang, Z.-H., … Yan, S. (2021). Tokens-to-token vit : Training vision transformers from scratch on imagenet. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 558–567).

Zador, P. (1963). *Development and evaluation of procedures for quantizing multivariate distributions* (Rapport technique). Stanford.

Zador, P. (1982). Asymptotic quantization error of continuous signals and the quantization dimension. *IEEE Transactions on Information Theory*, *28*(2), 139–149.

Zhang, B., Xiong, D., Xie, J., & Su, J. (2020). Neural machine translation with gru-gated attention model. *IEEE transactions on neural networks and learning systems*, *31*(11), 4688–4698.

Zhang, G., Patuwo, B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks : : The state of the art. *International journal of forecasting*, *14*(1), 35–62.

Zhang, X., Wu, L., & Chen, Z. (2022). Constructing long-short stock portfolio with a new listwise learn-to-rank algorithm. *Quantitative Finance*, *22*(2), 321–331.

Zhang, Y. (2004). *Prediction of financial time series with hidden markov models* (Thèse de doctorat non publiée). Applied Sciences : School of Computing Science.