Alma Mater Studiorum · Università di Bologna

Scuola di Scienze · Sede di Bologna
Dipartimento di Informatica - Scienza e Ingegneria

Tesi di Laurea Magistrale in
Finanza Computazionale

Deep Learning methods for Portfolio Optimization

Corso di Laurea Magistrale in
Informatica

Supervisor:
Prof.
FABRIZIO LILLO

Presented by:
FEDERICO BERTANI

I Appello - II Sessione
Anno Accademico 2020/2021

Abstract

Portfolio optimization is one of the most studied fields that have been researched with machine learning approaches because of its inherent demand for forecasting future market properties. In this thesis, it is shown how one can use deep neural networks with historical returns to do risk adjusted asset allocation. Unlike previous studies which set as target variable asset prices, the variable to predict here is represented by the best asset allocation strategy. Experiments performed on a time period of seven years show that temporal convolutional networks are superior to long short term memory networks and transformers. Compared to baseline benchmarks, the computed allocation has an average increase in the year revenue between 2% and 5%. Furthermore, results are compared against equally weighted, inverse volatility and risk parity methods, showing higher cumulative returns than the first method and equal if not higher in some cases than the latters methods.

L'ottimizzazione del portafoglio è uno dei campi più ricercati con approcci di *Machine Learning* a causa della sua domanda intrinseca di previsione delle proprietà future del mercato. In questa tesi, si mostra come si possono usare le *Deep Neural Networks* per fare *asset allocation* con una considerazione del rischio utilizzando i rendimenti storici. A differenza degli studi precedenti che fissano come variabile obiettivo i prezzi degli asset, qui la variabile da prevedere è rappresentata dalla migliore strategia di asset allocation. Gli esperimenti eseguiti su un periodo di tempo di sette anni mostrano che le *Temporal Convolutional Neural Networks* sono superiori alle *Long Short Term Memory Networks* e ai *Trasformers*. Rispetto ai benchmark di base, l'allocazione calcolata ha un aumento medio delle rendimenti annuali tra il 2% e il 5%. Inoltre, i risultati sono confrontati con altri vari metodi: allocazione equipesata, volatilità inversa e parità di rischio, mostrando rendimenti cumulativi superiori al primo metodo e uguali se non superiori in alcuni casi agli ultimi metodi.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis we research, find and describe a machine learning model for asset allocation. Asset allocation deals with the assignment of assets weights in an investment portfolio. This task and the finance studies field related to it is described in chapter 2. The allocation produced by the model has to respect several requirements: it has a minimum and maximum investment exposition for each asset and it has to allow the user to have flexibility in maximizing the cumulative return while still maintaining volatility under a certain limit. We compared the allocation generated by the model against several benchmarks: a baseline algorithm based on components past Sharpe Ratio, equally weighted, inverse volatility and risk parity allocation methods. We achieved a higher cumulative return of the baseline algorithm and equally weighted in every validation period, while equal or better return of equally weighted or inverse volatility methods. The code generating these results is available on GitHub [1].

A proper assignment of weights can decide the success or failure of an investment plan. It is a relevant topic of research and numerous related works are published annually. Its complexity is intrinsic in the understanding on which assets are better to invest, therefore giving more weight. But financial markets are very complex systems. Price time series are non-linear, non-stationary, chaotic and noisy. In the seventies, efficient market hypothesis (EMH) was established by initial work of *Malkiel and Fama (1970)* [2], according to which financial markets follow random pathways and therefore are unpredictable. In more recent work, like the one of *Kumar, Meghwani and Thakur (2016)* [3], there is evidence contrary to the efficiency of financial markets and the search for models and profitable systems is still attracting a lot of attention from academia. A predictive model

[1] `https://github.com/federicoB/Deep_Portfolio_Optimization`

capable of consistently generating returns above the market indices over time would represent strong evidence contrary to EMH. Fama himself, in a later work, revised his statement, indicating different levels of efficiency.

The challenges that a work like this thesis face are many: the selection of the best features to train the network, the selection of the best methods and architecture in a very extensive literature and making the network generalize better and perform well also out of sample.

As a machine learning model, we used neural networks, and specifically architectures made for sequence modelling tasks. Temporal Convolutional Networks (TCNs), Long Short Term Memory Networks (LSTMs) and Transformers were analyzed, used and compared. All these architectures layers and parameters are described in chapter 3 and more implementation details can be found in chapter 4. We found out that TCNs were superior to the other two architectures both in terms of allocation quality and computational speed. Allocation quality can be defined both in term of high Sharpe Ratio or high cumulative return, it depends what the investor is looking for.

Two different loss functions were analyzed, one inspired by the work of *Zhang* (2020) [4] computes the Sharpe Ratio of the portflio given the network output weights and aims to maximize it. This was very heavy computationally and did not showed relevant results so we moved to a second method. This second loss function is a mean squared error between the output weights and a target allocation generated a priori. This target allocation is generated knowing future returns and volatility in the training set. One important results of this thesis is that we show there is a correlation between target allocation parameters, i.e. return, Sharpe ratio and standard deviation, and the same parameters in the generated allocation. This means the network is learning to generate an allocation with the same characteristics of the allocation it was trained with. Furthermore and most important, this allow the user of the model to control risk.

This work was done in collaboration with Salzenberg AI, an fintech company which provides trading signals for a managed investment fund. The fund trades four futures based on the Nasdaq100, Sp500, EuroSTOXX50 and Nikkei225 indices. My collaboration and internship was the search for a method based on machine

learning for the allocation of capital between the four different assets. The model is not trained on returns of the Futures, but on *strategy returns* over them. We create strategy returns for each of the 4 assets by multiplying the returns of a linked future with the values of a trading signal (see section 2.12). The trading signal is provided by Salzenberg AI, and is based on a proprietary strategy. Data available span from 2013 to 2020.

The introduction continues doing a literature review on the subject. Chapter 2 and 3 cover the background knowledge behind this work, on finance and machine learning respectively. Chapter 4 delineates the methodology followed in this work and how we avoided some problems. Chapter 5 shows all the results of the features and model selection. In the end, Chapter 6 recaps all the thesis work and describe possible future improvements.

1.1 Literature review

We introduce and briefly describe several recent works on the field of machine learning applied to asset allocation. This literature review by no means want to be exhaustive but want to give to the reader an idea of the different approaches that the research is following for solving this task.
The first work that has to be cited is the one of *Snow (2020)* [5] as it is not directly an experiment but a formalization of the problem of asset allocation in many machine learning methods including regressions, autoencoders and reinforcement learning. Even if there are no experimental results but the formalization is clear and useful for a mathematical introduction in the field of machine learning for portfolio optimization.

We then found other five works than can be subdivided either for the data used or the method applied. Three out of the five works use only historical assets prices like this thesis, while another one combined them with macro-economics indicators and another one with sentiments data.
Regarding the methods used, only one work tries to use Random Forest, two works use a Multi Layer Perceptron, two use Long-Short Term Memory Networks and other two use Reinforcement Learning.

We first have to cite the work of *Zhang et al. (2020)* [6] as it was influential for this thesis and was the base from which we started.

Zhang developed a model that has the Sharpe ratio (see chapter 2) as objective function to find the optimal portfolio weights using a LSTM Neural Network.
As said by *Tütüncü (2006)* [7] optimizing Sharpe ratio by traditional means of quadratic programming is somewhat complicated as it is a non convex optimization problem. The traditional way would be reducing it to an equivalent problem, but machine learning can offer an alternative solution.

An interesting aspect of a machine learning solution to portfolio optimization is that it bypasses the step of forecasting the returns which are needed to apply Markowitz model.
Several works ([4,8,9]) argue that the return forecasting methods does not guarantee that a portfolio's return will be maximized since the network optimizes a function targeting a more precise prediction on next day prices not next day portfolio performance.

With this neural method, expected returns are not necessary, portfolio weights are computed directly from historical returns. This can be considered as an easier task for a neural network as it does not have to precisely predict the next daily price but an *ordering* between the assets. Still, the quantitative difference between next day assets returns has to be implicitly predicted to decide how much to allocate on each.

The architecture that Zhang uses is an LSTM with 50 units, followed by a fully connected layer and a softmax. The indices used are: US total stock index (VTI), US aggregate bond index (AGG), US commodity index (DBC) and Volatility Index (VIX). Results show as their model delivers the good performance in the testing period 2011-2020 as can be seen in figure 1.1.
A sensitive analysis is included to understand how input data contributes to outputs and the observation meet the econometric understating that most recent data is most relevant. Still this could be consequence of LSTM having problem to underline long term dependencies [1].

Now two works that use Reinforcement Learning (RF). In finance, RF is particularly interesting as it tries to mimic human behaviour of searching for a reward and it is well know that the complex phenomenons observable in a financial markets are all the sum of many actors each trying to maximize their gains. Therefore, Reinforcement Learning could offer an appropriate way to model these agents

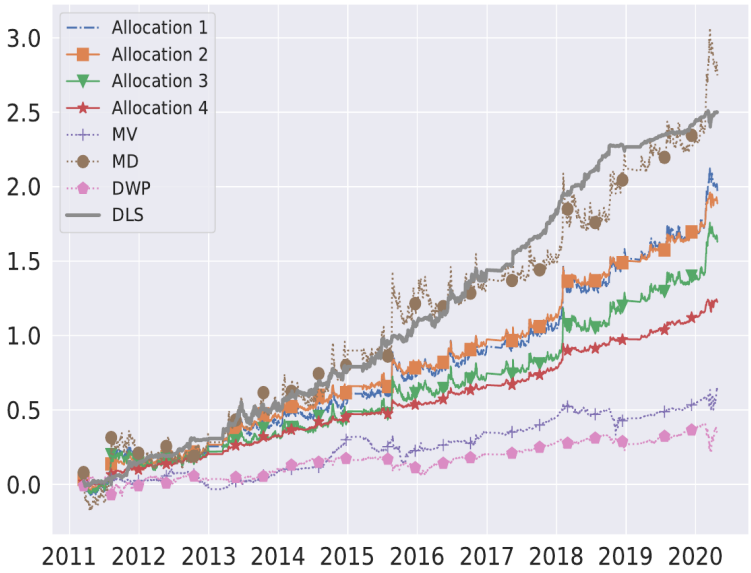


Figure 1.1: **Zhang et al. (2020) results**: cumulative returns. Allocation 1 is equal allocation, Allocation 2 is 50-10-20-20, Allocation 3 is 10-50-20-20, Allocation 4 is 40-40-10-10, MV means Mean-Variance optimization, MD means Maximum Diversification, DWP means Diversity weighted portoflio from Stochastic Portfolio Theory, DLS is Zhang model

behaviour.

Wijs (2018) [10] compare its reinforcement learning approach to portfolio optimisation with the one of *Campbell (2002)* [11] that use a Vector Autoregressive (VAR) model to forecast future returns. Wijs use 3 American assets: 3-month Treasure Bill, 5-year Treasury-Note and the weighted average of NYSE, NASDAQ and AMEX. Data spans a large period from 1954 to 2016. Results show a cumulative return of 33% compared to Campbell method, a reduction of volatility of 33% and a 3% lower turnover.

Kim (2020) [12] is a very interesting work as it combines Reinforcement learning with Transformers. The reward is risk-adjusted as it includes the Sortino ratio. The agent uses a deep neural network, including Transformers layers, as a policy approximator. The model is trained and evaluated with assets of nine Dow Jones companies representing each sector. Data spans 20 years, from 2000 to 2020. The model takes sequences of 50 days. Results show a cumulative return over the years 2018 to 2020 of 43% and an annualized Sharpe ratio of 0.64. Figure 1.2 and table 1.1 summarize these results.

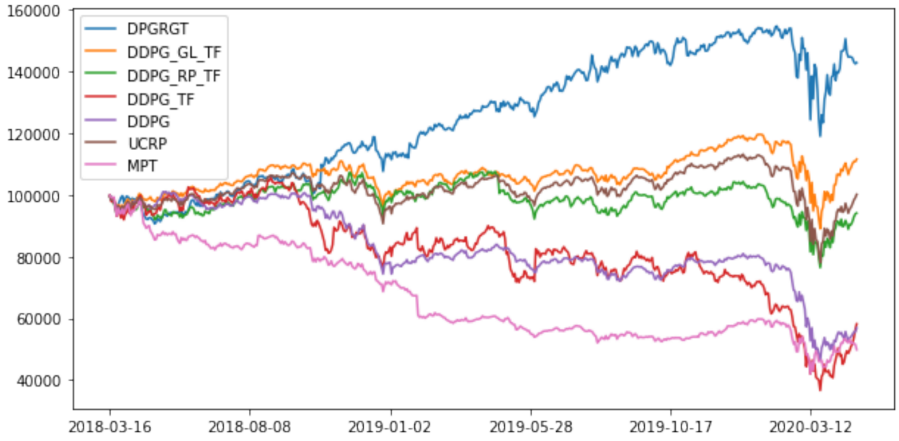


Figure 1.2: **Kim (2020) results**: portfolio value after initial value of 10 thousand. MPT is Markowitz's Modern Portfolio Theory, UCRP is Uniform Constant Rebalanced Portfolio, DDPG stands from Deep Deterministic Policy Gradient and DDPG_GL_TF, DDPG_RP_TF, DDPG_TF are DDPG with Gated Transformer, Relative Attentional Transformer and standard Transformer respectively

We now show two works where not only historical prices are used but macroeconomics indicator and sentiment analysis are added to training data. While this makes the comparison with the previous works and the works in this thesis more difficult, it is interesting to examine these results as they can give an insight on what is can be reached with more type of data available.

Chakravorty et al. (2018) [13] used price-volume together with macroeconomics data in a feed-forward shrinking architecture for asset allocation. They use lagged returns over [260, 130, 60, 30] days partly to eliminate short term noise from the daily return and partly to keep turnover of the resulting investment strategy low. The macro economics data used are US bond spread, Gold, Copper and Oil prices, US inflation, US non-farm job creation and US GDP growth. Two models were researched, one that forecasting expected returns and one directly computing weight by optimizing Sharpe ratio. The results performance are comparable to risk parity method (see Chapter 2)

Malandri et. al (2018) [14] add to historical price data, financial sentiment collected from Twitter, gaining better results than using only historical prices. It uses this data for training a model optimizing asset allocation in the New York Stock

Model	Cumulative return (%)	Annualized Sharpe Ratio
DPGRGT	43.16	0.6418
DDPG_GL_TF	11.93	0.2813
DDPG_RP_TF	-5.45	-0.1343
DDPG_TF	-41.71	-0.8191
DDPG	-42.91	-1.2194
UCRP	0.53	0.0125
MPT	-50.07	-1.5840

Table 1.1: **Kim (2020) results**: cumulative return and annualized Sharpe Ratio. MPT is Markowitz's Modern Portfolio Theory, UCRP is Uniform Constant Rebalanced Portfolio, DDPG stands from Deep Deterministic Policy Gradient and DDPG_GL_TF, DDPG_RP_TF, DDPG_TF are DDPG with Gated Transformer, Relative Attentional Transformer and standard Transformer respectively

Exchange (NYSE). Historical prices are from 15 popular stocks. Sentiment data is the number of positive, negative, neutral comments and their daily change. They compare three different architectures: Multi layer perceptron, Random Forest and Long Short Term Memory network. Allocation is directly found by the network in an end-to-end fashion. The network is trained matching an allocation that allocates 100% on the most remunerative asset. Finding shows how LSTM is superior with respect to the other two architectures. Unfortunately, the allocation performance is only compared against equally weighted portfolio. Still, it scores an increase in cumulative yearly return of 5% without sentiment data and 19% with.

Chapter 2

Portfolio optimization

This chapter deals with the financial background necessary to understand the remaining of this thesis. It is targeted to readers that have already knowledge of what an investment asset is. Investments assets have many typologies and covering all would be out of scope of this work. It is sufficient to say that all of them has a price, fundamental quantity from which return and volatility are derived. Price is influenced by many factors, but in origin, how much investors have trust in that asset. This chapter deals more with *how* to combine assets to form portfolio to reach investment goals. After the definition of assets and portfolio we explain diversification, a method to reduce investment risk. Then the mean-variance diagram is introduced, a way to visualize the different strategies for a portfolio construction. Subsequently, we cover different popular asset allocation methods, as this thesis create a new one of them in the later chapters.

2.1 Asset return

An investment instrument that can be bough and sold is often called an **asset**. Typical examples of assets are stocks, bonds, ETFs and futures. An asset sold in a market has an **opening price**, the price of the asset when the market opens, and a **closing price**, the price of the asset when the market closes. There are two important definitions to give: **total return** of an asset and **rate of return** of an asset, respectively indicated usually with R for the former and r for the latter.

$$\text{total return} = R = \frac{\text{closing price}}{\text{opening price}}$$

$$\text{rate of return} = r = \frac{\text{closing price - opening price}}{\text{closing price}}$$

The two notions are related by:

$$R = 1 + r$$

Frequently the shorter expression *return* means the rate of return. Because of after-hours trading, close price of day t and opening price of day $t + 1$ can differ. A **cumulative return** on an investment is the total amount gained or lost by the investment throughout time, regardless of the length of time involved. Given a sequence of rate of returns r_i for $i = 1, ..., n$ their cumulative return is defined as

$$C_r = \left(\prod_{i=1}^{n} 1 + r_i \right) - 1$$

Short selling or **shorting** is a trading or investing technique that looks for a profit on a stock's or other security's price declining. To do this, an asset is borrowed by someone who owns it, such a brokerage firm. The borrowed asset is then sold to someone else, receiving an amount X_0. At a later date, you repay the loan by purchasing the same asset for an amount X_1 and return the asset to your lender. If the amount X_1 is lower than the original amount X_0, a profit of $X_0 - X_1$ is made. Therefore, short selling is profitable if the asset price declines.

Many investors regard short selling as extremely risky, if not dangerous. The reason for this is because there is no limit to the amount of money that can be lost. If the asset value increases, the loss is $X_1 - X_0$, since X_1 can increase arbitrary, so can the loss. For this reason short selling is prohibited within certain financial institutions and it purposely avoided as a policy by many individuals and institutions. In a long position, the opposite of short selling, the loss is limited by the amount invested. If we buy at price X_0 and sell at price X_1, the worst that can happen is that $X_1 = 0$ with a loss $L = X_0$, that would mean we lost (only) all the money we invested. Short selling is not prohibited everywhere, and there is a significant amount of short selling of stock market assets. Let us determine the return associated with short selling. We receive X_0 initially and pay X_1 later, so to expenditure is $-X_0$ and the final reception is $-X_1$ and therefore return is:

$$R = \frac{-X_1}{-X_0} = \frac{X_1}{X_0}$$

The minus signs cancel out, so we obtain the same expression as using long positions. As a result, the return value R applies algebraically to both long and short positions.

Leverage is when we use borrowed capital to increase the potential return of an investment. The result is to multiply the potential returns from an investment. At the same time, leverage will also multiply the potential loss in case the investment does not succeed. Leverage always have a multiplier, called *leverage multiplier*, that we are going now to indicate with m. Let us suppose we buy an asset daily leveraged at price X_0 and multiplier m, what we are actually doing is borrowing $(m-1)X_0$ capital to buy mX_0 amount of that asset. At the end of the day it will be sold for price mX_1 but our initial expenditure was only X_0 with an actual total return of:

$$\text{leveraged total return} = m\frac{X_1}{X_0}$$

The borrowed capital has then to be lend back but still as a result, our return is multiplied by factor m increasing both the eventual profit or loss.

2.2 Portfolios

An investment portfolio is a collection of financial investments, also called assets. In some literature portfolios are also called *master assets*.
Let us suppose we have n available assets. We form the portfolio by investing a specific amount in each asset. We indicate the amount invested in asset i for $i = 1, 2, ..., n$ as X_{0i} such that $\sum_{i=1}^{n} X_{0i} = X_0$. If we are allowed to sell and asset short, then some of the X_{0i}'s can be negative, otherwise we restring X_{0i}^s to be non negative. The amounts invested can be expressed as fractions of the total investment. Therefore we write:

$$X_{0i} = w_i X_0,\ i = 1, 2, ..., n$$

where w_i is the **weight** or fraction of asset i in the portfolio. Clearly:

$$\sum_{i=1}^{n} w_i = 1$$

and some w_i's can be negative is short selling is allowed. Let R_i denote the total return of asset i. Then the amount of money generated at the end of the period by the ith asset is $R_i X_{0i} = R_i w_i X_0$. The total amount received by this portfolio at the end of the period is therefore $\sum_{i=1}^{n} R_i w_i X_0$. As a result we find that the overall total return of the portfolio is the weighted sum of the returns of its components:

$$R = \frac{\sum_{i=1}^{n} R_i w_i X_0}{X_0} = \sum_{i=1}^{n} w_i R_i$$

Equivalently, since $\sum_{i=1}^{n} w_i = 1$, we have: $r = \sum_{i=1}^{n} w_i r_i$

2.3 Asset volatility

Since the money obtained when selling an asset is uncertain at the time of purchase the return is unknown and random. Therefore it can be described in probability theory. Many concept of probability are used in finance but here we will introduce only variance.

We use the notation $\mathbb{E}$ to indicate the expected value of a random variable x. For convenience $E(x)$ is often denoted by $\bar{x}$. Also the terms **mean** or **mean value** are often used for the expected value.

The expected value of a random variable provides a useful summary of the probabilistic nature of the variable. However, typically one wants, in addition, to have a measure of the degree of possible deviation from the mean. One such measure is the **variance**. Given a random variable y with expected value $\bar{y}$, the quantity $y - \bar{y}$ is itself random, but has an expected value of zero. This is because $\mathbb{E}(y - \bar{y}) = \mathbb{E}(y) - \mathbb{E}(\bar{y}) = \bar{y} - \bar{y} = 0$. The quantity $(y - \bar{y})^2$ is always non negative and is large when y deviates greatly from $\bar{y}$ and small when it is near $\bar{y}$. The expected value of this squared variable $(y - \bar{y})^2$ is useful measure of how much y tends to vary from its expected value. In general, for any random variable y the variance of y is defined as:

$$var(y) = \mathbb{E}[(y - \bar{y})^2)]$$

In mathematical expressions, variance is represented by the symbol σ^2. Therefore we write $\sigma_y^2 = var(y)$ or if y can be deduced by context, we simply write $\sigma^2 = var(y)$.

We frequently use the square root of the variance, denoted by σ and called the **standard deviation**. It has the same units as the quantity y and is another measure of how much the variable is likely to deviate from its expected value. Formally:

$$\sigma_y = \sqrt{\mathbb{E}[(y - \bar{y})^2]}$$

It is good to know, as it is useful in computations that:

$$\begin{aligned} var(x) &= \mathbb{E}[(x - \bar{x})^2] \\ &= \mathbb{E}(x^2) - 2\,\mathbb{E}(x)\bar{x} + \bar{x}^2 \\ &= \mathbb{E}(x^2) - \bar{x}^2 \end{aligned} \tag{2.1}$$

In finance, the term **volatility** is often used either to indicate asset variance or asset standard deviation.

2.4 Portfolio covariance

When considering two or more random variables, as assets returns in a portfolio, their mutual dependence can be summarized conveniently by their **covariance**.

Let x_1 and x_2 be two asset returns with expected returns $\bar{x_1}$ and $\bar{x_2}$. The covariance of these assets is defined as:

$$cov(x_1, x_2) = \mathbb{E}[(x_1 - \bar{x_1})(x_2 - \bar{x_2})]$$

The covariance of two assets x and y is frequently denoted by σ_{xy}. Consequently, for assets x_1 and x_2 we write $cov(x_1, x_2) = \sigma_{x_1,x_2}$ or alternatively, $cov(x_1, x_2) = \sigma_{12}$. Note that, by symmetry, $\sigma_{12} = \sigma_{21}$ Analogous to equation 2.1 there is an alternative shorted formula for covariance that is easily derived:

$$cov(x_1, x_2) = \mathbb{E}(x_1 x_2) - \bar{x_1}\bar{x_2}$$

This is useful in computations. If two assets x_1 and x_2 have the property that $\sigma_{12} = 0$, then they are said to be **uncorrelated**. Beware that if two assets are uncorrelated it does not imply that they are independent, the situation where knowledge of the return of one asset gives no information about the other. Correlation measure linear association but if the two assets are related in other non-linear ways correlation could not distinguish from independent case.

A portfolio can be composed of different assets with different return-risk ratio characteristics. An adequate balance between this can allow the investor to fit its preferences.

2.5 Diversification

We now determine the variance of the rate of return of a portfolio. We denote the variance of the return of asset i by σ_i^2, the variance of the rate of return of the

portfolio by σ^2, and the covariance of the return of asset i with asset j by σ_{ij}^2:

$$\begin{aligned}
\sigma^2 &= \mathbb{E}[(r-\bar{r})^2] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{n} w_i r_i - \sum_{i=1}^{n} w_i \bar{r_i}\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{n} w_i (r_i - \bar{r_i})\right)\left(\sum_{j=1}^{n} w_j (r_j - \bar{r_j})\right)\right] \\
&= \mathbb{E}\left[\sum_{i,j=1}^{n} w_i w_j (r_i - \bar{r_i})(r_j - \bar{r_j})\right] \\
&= \sum_{i,j=1}^{n} w_i w_j \sigma_{ij}
\end{aligned} \tag{2.2}$$

This important result shows how the variance of a portfolio return can be calculated easily from the covariances of the pairs of asset returns and the asset weights used in the portfolio.

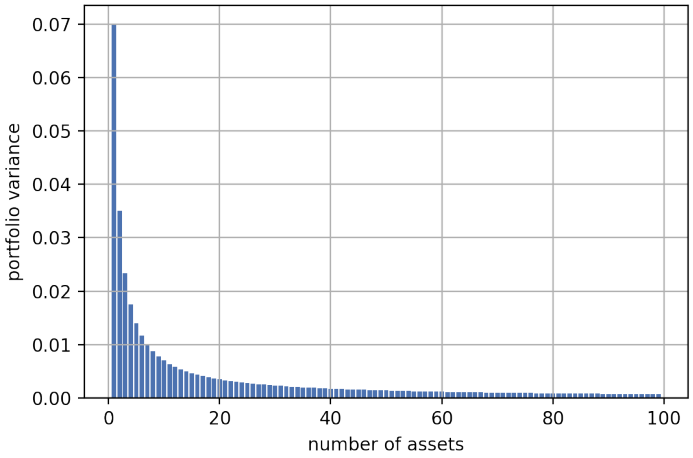
Portfolio with only a few assets may be subject to a high degree of risk, represented by a relatively large variance. As a general rule, the variance of the return of a portfolio can be reduced by including additional assets in the portfolio, a process referred to as **diversification**.
This can be shown by using the following formulas. Let us suppose an example with n assets, all of which are mutually uncorrelated, that means each asset is uncorrelated with any other asset in the group. Let us suppose also that the rate of return of each asset has mean m and variance σ^2. A portfolio is built by taking an equal portion of the n assets, that means $w_i = \frac{1}{n} \forall i$. The total rate of return of this portfolio is then:

$$r = \frac{1}{n} \sum_{i=1}^{n} r_i$$

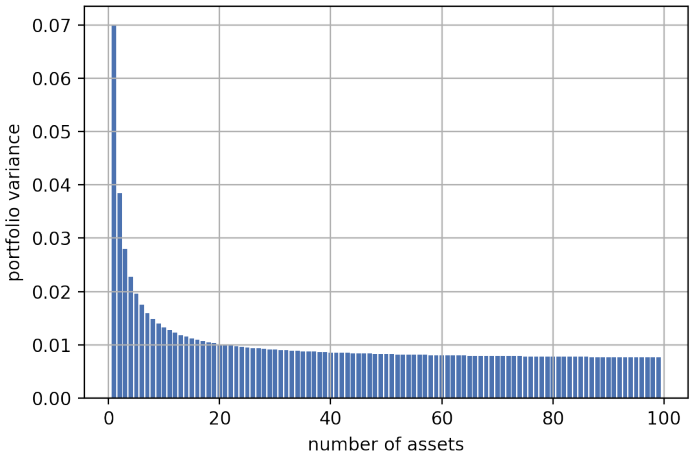
But since each asset has mean rate of return m also the mean value of r is independent of n.
The variance of r is:

$$\sigma(r) = \frac{1}{n^2} \sum_{i=1}^{n} \sigma^2 = \frac{\sigma^2}{n}$$

The variance decrease rapidly as n increases, as shown in figure 2.1.



(a) Uncorrelated assets $\sigma^2 = 0.07$



(b) Correlated assets $\sigma^2 = 0.07$
$cov(r_i, r_j) = \frac{1}{10}\sigma^2$

Figure 2.1: **Effects of diversification** If assets are uncorrelated variance of portfolio rapidly decrease and tends to zero by increasing number of assets. If assets are correlated there is a lower limit to variance.

This is possible because we assumed that the individual returns are uncorrelated. If the asset are uncorrelated, then the variance limit to is zero:

$$\lim_{n\to\infty} \frac{\sigma^2}{n} = 0$$

If the assets were even partially correlated, then by adding more assets the variance would decrease slower.
We now prove this. Let us suppose that each asset has a rate of return with mean m and variance σ^2, but now each return pair is correlated by a factor t and has then covariance $cov(r_i, r_j) = t\sigma^2$ for $i \neq j$. The portfolio is formed by equal portions of n of these assets. In this case:

$$\begin{aligned}
var(r) &= \mathbb{E}\left[\sum_{i=1}^{n} \frac{1}{n}(r_i - \bar{r})\right]^2 \\
&= \frac{1}{n^2}\,\mathbb{E}\left\{\left[\sum_{i=1}^{n}(r_i - \bar{r})\right]\left[\sum_{j=1}^{n}(r_j - \bar{r})\right]\right\} \\
&= \frac{1}{n^2}\sum_{i,j}\sigma_{ij} = \frac{1}{n^2}\left(\sum_{i=j}\sigma_{ij} + \sum_{i\neq j}\sigma_{ij}\right) \\
&= \frac{1}{n^2}[n\sigma^2 + t(n^2 - n)\sigma^2] \\
&= \frac{\sigma^2}{n} + t\sigma^2(1 - \frac{1}{n}) \\
&= \frac{\sigma^2(1-t)}{n} + t\sigma^2
\end{aligned} \tag{2.3}$$

This results is shown in figure 2.1(b). In this case it is impossible to reduce the variance below $t\sigma^2$, no matter how large is n

$$\lim_{n\to\infty} \frac{\sigma^2(1-t)}{n} + t\sigma^2 = t\sigma^2$$

In the previous analysis we assumed that the expected retun of all assets are equal. This is not, in general, the case. Diversification may reduce the overall expected return while reducing the variance. Most people are averse to reducing a considerate amount of return in exchange for a little reduction in variance, so diversification is not a task that can be executed blindly.

2.6 Mean-Variance Diagram

The random rates of return of assets can be expressed on a two-dimensional diagram, as shown in figure 2.2 . As asset with mean rate of return $\bar{r}$ and standard deviation σ is represented as a point in this diagram. The standard deviation is plotted on the horizontal axis, and the vertical axis is used for the mean. This diagram is known as the **mean-standard deviation diagram**, or simply $\bar{r} - \sigma$ diagram.

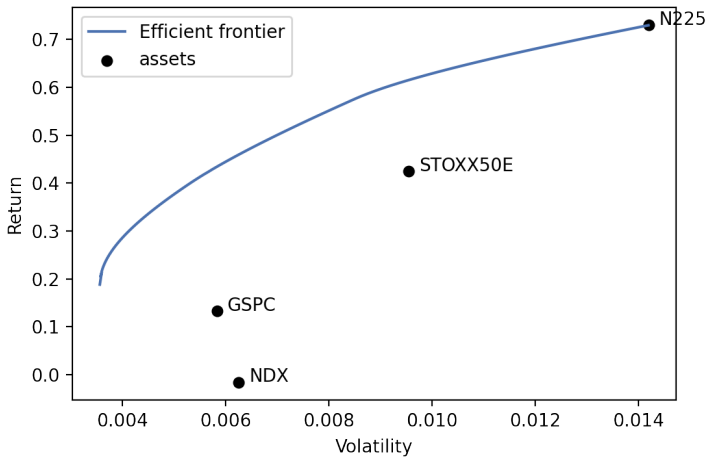


Figure 2.2: Mean-Variance diagram: Points are the four different assets obtained from strategy signals and index futures used in this work, the curve is the efficient frontier of portfolios created with these assets.

Given n assets on a mean-standard deviation 2D plot, where they are points, these assets can be combined to form a portfolio that will have a mean return and standard deviation itself, thus represented itself as a point in the diagram. The new portfolio mean return and standard deviation can be calculated from the assets means, variances and covariances of returns of the original assets. However, since covariances are not shown on the diagram, the exact location of the point representing the new asset cannot be determined from the location on the diagram of the original assets. There are many possibilities, depending on the covariance of the asset returns. All these possibilities of the location of the compound portfolio in the diagram form a region, called **feasible region** or **feasible set** of points. With only two assets the region will be a curve, with at least three assets the region will be a two-dimensional solid region.

The feasible region is convex to the left, that means that given two random points in the feasible region, the line connecting them does not cross the left boundary of the feasible region. This is because all portfolios with positive weight made from two assets lie on the left of the line connecting them. This is because the new portfolio will have always a mean return higher than the lowest mean return of the assets and a variance always lower than the highest of the assets. If short selling is allowed, so the weights can be negative, the region will always contain the region where short selling is not allowed, as see in figure 6.2 in the appendix.

The left boundary of a feasible region is called the **minimum variance region**, since for any value of the mean rate of return, defining a horizontal line, the point will be the one with the smallest variance staying in the leftmost point of the line. The minimum variance region has a characteristic bullet shape. There is a specific point in the minimum variance region that has a global minimum variance and is called the **minimum variance point**. An investor that given a fixed return would always prefer the leftmost point of minimum variance is called **risk adverse investor**, and an investor that would select a point other than the one of minimum standard deviation is labelled **risk preferring investor**. We can turn the previous analysis by 90 degree and consider portfolio on a vertical line, that means portfolio with a fixed standard volatility. Investors in this case would always prefer the highest point, in others words the one with the highest return. This property of investor is called **nonsatiation**. This implies that only the upper part of the minimum variance region will be the interest of investors who are risk adverse and non satiated. The upper part of the minimum variance region is called **efficient frontier**.

2.7 Sharpe ratio

Sharpe ratio was created by Nobel Laureate William F. Sharpe and is used to help investors understand an investment's return against its risk. The ratio is the average return earned in excess of the risk-free rate per unit of volatility or total risk.
In general, the higher the Sharpe ratio, the more attractive the risk-adjusted return of the investment.
An investor will better isolate the profits associated with risk-taking operations by subtracting the risk-free rate from the mean return. The risk-free rate of return is the return on a risk-free investment, i.e., the return investors would expect if they

take no risk. The risk-free rate, for example, may be the yield on a US Treasury bond.

$$S_r = \frac{r - r_f}{\sigma_r}$$

r rate of return
r_f Risk free return
σ_r Standard deviation of returns
But the Sharpe ratio also have several limitations:

- It uses standard deviation as a measure of risk. This would imply that returns are normally distributed, but financial markets shows large number of surprising drops or spikes in prices.
- It focuses on volatility but not its direction. It cannot distinguish between upside and downside trends. Rare events of large positive returns should be considered beneficial, while rare large losses should be considered the opposite. The Sharpe ratio considers the two tail of the distribution the same.
- As with most metrics and parameters, Sharpe ratios use historical returns and volatility. The decisions based on the ratio assume future performance will be similar to the past, that is often not true.

The conversion of a short-term estimate or rate into an annual rate is known as **annualization**. An investment with a short-term rate of return is typically annualized to get an annual rate of return, which may involve compounding or reinvestment of interest and dividends. It is useful to annualize a rate of return in order to compare one security's performance to that of another. To annualize the Sharpe ratio computed on daily returns over a period of t days:

$$S_{rA} = \frac{(R - R_f) \cdot \frac{252}{t}}{\sigma_r \cdot \sqrt{252}} = S_r \cdot \frac{\sqrt{252}}{t}$$

The reason behind this is because the returns of the portfolio are diffusive, as in a Wiener process, in which volatility scales with the square-root of time. Also, in average, there are 252 trading days in a year.

2.8 Post-Modern Portfolio theory

As explained above, variance is not always a the best measure of risk and Modern Portfolio theory has many critics. Post-Modern Portfolio theory (PMP) has been

created [15]. Markowitz itself suggested that a model based on semi-variance would be preferable, but it becomes a harder computational problem. PMP theory only use semivariance, we now give a definition of it.
The first step of calculating the semivariance is to choose a minimum acceptable return (MAR). Popular choices include zero and the risk-free rate for the year. We will use zero here. Secondly, we select only the returns that are lower or equal than the MAR. Finally the variance of this selected returns is computed. Formally:

$$\text{semivariance} = SV(r) = \sum_{i=1}^{n} (\mathbb{E}[(r_i - \bar{r})^2 1_{r_i \leq 0})$$

where $1_{r \leq 0}$ is and indicator function, i.e.

$$1_{r \leq 0} = \begin{cases} 1 & if\ r \leq 0 \\ 0 & else \end{cases}$$

The Sortino ratio was created to address the Sharpe ratio's inability to differentiate between types of risk. It only uses downside risk, nominally the square root of semivariance.

$$\text{Sortino ratio} = \frac{r - r_f}{\sqrt{SV(r)}}$$

In a bull market we should seek for as much volatility as possible, only in a bear market volatility should be avoided. Individuals are more concerned with avoiding loss that seeking gains. From a practical standpoint, risk is not symmetrical. Still, in this work it is going to be used mainly Markowitz initial theory that we consider sufficient for our objectives.

2.9 Asset allocation methods

Asset allocation is that part of investment science related to find an optimal balance between risk and return of a portfolio by determining or changing portfolio components weights. Rebalance of portfolio follows consideration of investor risk tolerance and investment horizon.
Markowitz model is an asset allocation technique that aims to select the best asset distribution within a portfolio in order to maximize returns at a given risk level. Markowitz's work is widely known as modern portfolio theory [16]. Markowitz problem is practically the problem of finding points on the efficient frontier. Assuming n assets each with mean (or expected) rates of return $\bar{r}_1, \bar{r}_2, ..., \bar{r}_n$ and covariances σ_{ij} for $i, j = 1, 2, ...n$. A portfolio is defined by a set of n weights

w_i for $i = 1, 2, ..., n$ that sum up 1. Negative weights corresponds to short selling. To find a minimum variance portfolio, we set the mean value at some arbitrary value $\bar{r}$. Then we find the portfolio of minimum variance that has this mean. We formulate the problem as:

$$\begin{aligned} \text{minimize} & \frac{1}{2} \sum_{j=1}^{n} w_i w_j \sigma_{ij} \\ \text{subject to} & \sum_{i=1}^{n} w_i \bar{r_i} = \bar{r} \\ & \sum_{i=1}^{n} w_i = 1 \end{aligned}$$

Once the Markowitz problem is formulated, it can be solved both analytically and numerically to obtain a specific solution. Usually the analytical process is the most used.

Markowitz solution makes the mean and variance trade off explicit. This is useful for investors because, as seen in section 2.5 increasing the number of assets can reduce variance, and therefore risk, but could also reduce return. Not all investors would been keen to that. Markowitz solution allows to create portfolio suitable to different demands.

Predicting returns are required for mean-variance optimization and this is its primary difficulty even if the theory itself gives strong mathematical guarantees. In practice, determining returns with a sufficient degree of accuracy is challenging and as a result, the best we can do is often extrapolating them from historical data.

There are many common algorithms for portfolio optimization and asset allocation to consider as benchmark when developing a new method, as in this case. One of the most basic ones is to give to each asset an **equal weight**, while this can be seen as very rudimental it has been observed that still can result in good performance in validation datasets [17] [18] [19].

A second allocation method is the **inverse volatility** method also called inverse-variance weighting. In this method every asset receive a weight proportional to the inverse of its volatility. Formally:

$$w_i = \frac{\frac{1}{\sigma_i}}{\sum_{j=0}^{n} \frac{1}{\sigma_j}}$$

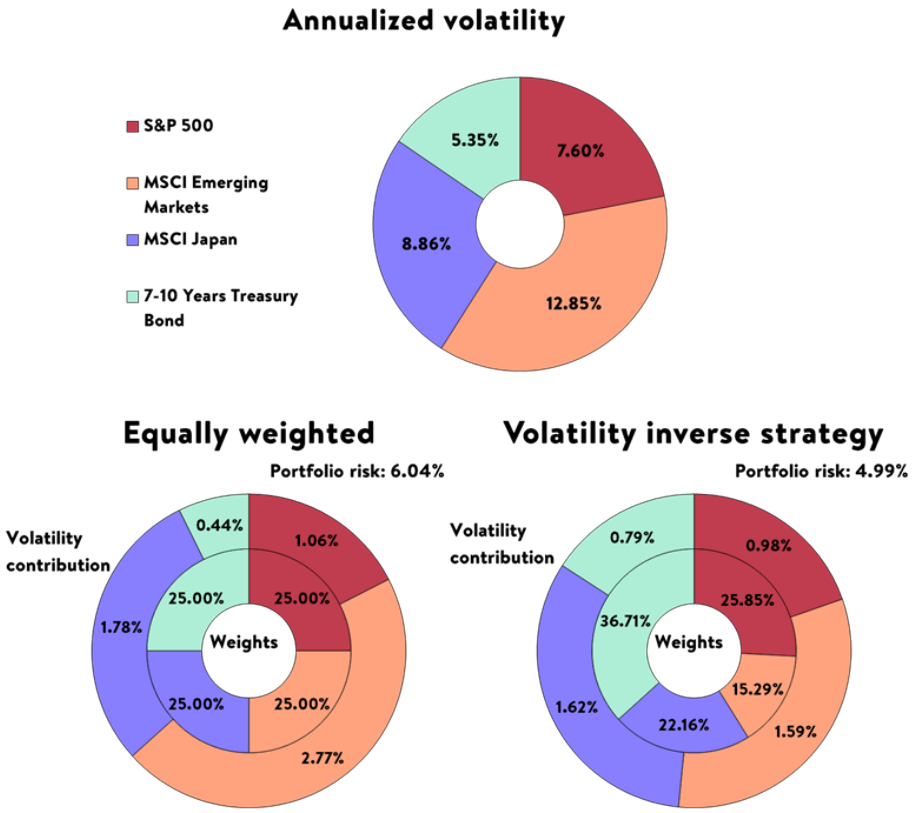


Figure 2.3: Graphical explanation of inverse volatility allocation method [20].

Another allocation method is the **risk parity** method. The objective of this method is to create a portfolio where each asset contributes equally to the portfolio overall volatility. According to the risk parity method, when asset allocations are modified (leveraged or deleveraged) to the same risk level, the risk parity portfolio can produce a higher Sharpe ratio and be more robust to market downturns than a standard portfolio [21]. Consider a portfolio of n assets with rate of returns $r_1, ..., r_n$ where the weights of the assets with rate of return r_i is w_i. The w_i form the allocation vector w. Let us further denote the covariance matrix of the assets by Σ. The volatility of the portfolio is then defined as the standard deviation of

the random variable $w_t X$ which is:

$$\sigma(w) = \sqrt{w' \Sigma w}$$

A risk parity portfolio can be obtained by solving the following minimization problem:

$$\arg\min_{w} \sum_{i=1}^{n} \left(\frac{\sqrt{w^T \Sigma w}}{n} - w_i \cdot c(w)_i \right)^2$$

Where $c(w)$ is the vector of marginal contribution to volatility of each asset ($\delta_{w_i} \sigma(w)$) and is computed as follows:

$$c(w) = \frac{\Sigma w}{\sqrt{w' \sigma w}}$$

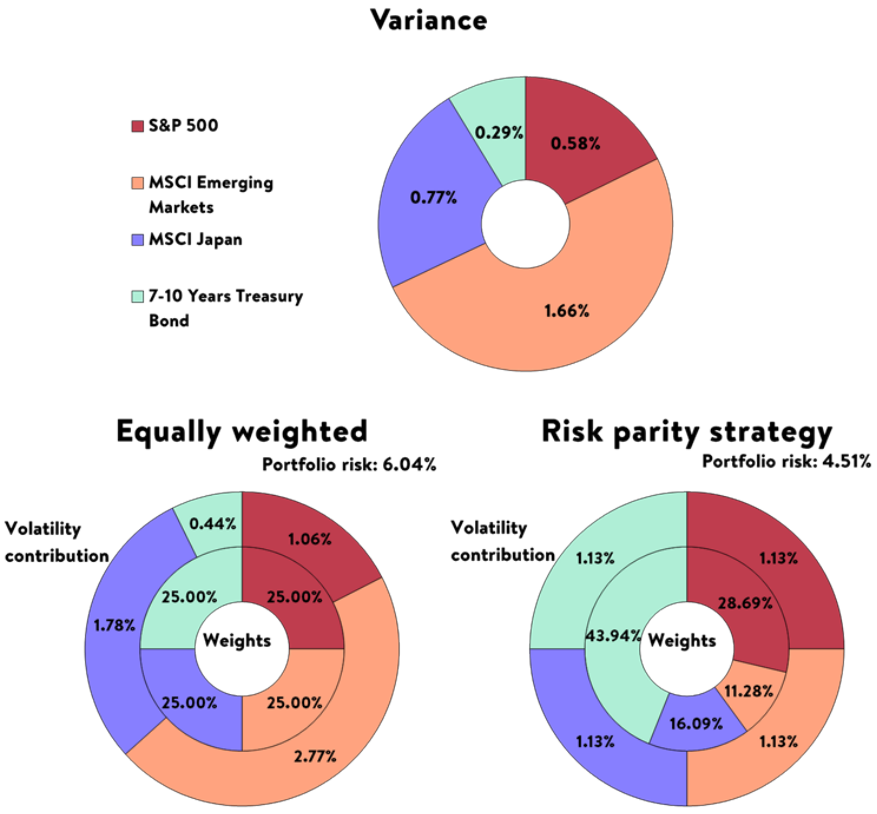


Figure 2.4: Graphical explanation of risk parity allocation method [20].

2.10 Relative strength index

Technical indicators are functions of financial quantities, generally prices or returns. They are helpful for technical analyst to forecast movements of future prices.

One of the most used technical indicators is the **Relative Strength Index (RSI)**. The RSI is a momentum oscillator that measures price movement velocity and magnitude. The momentum is the rate at which a price rises or falls [22]. The RSI is most commonly utilized over a 14 day period and it can have a value from 0 to 100, with 70 and 30 being the most common high and low points. For alternative shorter and longer outlooks, different size of time frames are used.
Each trading period is classified as up or down period. Up periods are characterized by the close being higher that the previous close. A down period is characterized by the close being lower than the previous period's close. We calculate U (Upward change) and D (Downward change) as the follows for an up period:

$$U = \text{close}_{\text{now}} - \text{close}_{\text{previous}}$$

$$D = 0$$

Instead, for a down period the following formula is applied:

$$U = 0$$

$$D = \text{close}_{\text{previous}} - \text{close}_{\text{now}}$$

The average U and D over multiple are calculated using a n-period smoothed **(SMMA)**, modified (MMA) or exponential (EMA) moving average.
The *relative strength factors* is the ratio of these averages:

$$RS = \frac{SMMA(U, n)}{SMMA(D, n)}$$

The relative strength factors is then normalized between 0 and 100 to obtain the relative strength index:

$$RSI = 100 - \frac{100}{1 + RS}$$

Many other technical indicators exists, like Moving Average Convergence/Divergence (MACD) [23], Bollinger bands [24], Fibonacci retracement [25], Ichimoku cloud [26], and Average directional movement index [22]. However, due to time constraints, we were unable to test the effectiveness of using these indicators applied to the input data. The RSI was chosen as it is one of the most popular technical indicators.

2.11 Volumes of trade

Another financial quantities sometimes used for forecasting financial quantities are trade volumes, the total number of shares that was traded during a given period of time. While volumes have low correlation with stock returns, they have a higher correlation with volatility and can be used to forecast it. One of the earlier studies on this correlation was the work of Schwert(1989) [27] e Gallant *et al.* (1992) [28].

2.12 Trade signal

A trade signal is a time series indicator of suggested selling or buying of a specific security, typical with a value of -1 or 1. Negative value means selling, while positive values means buying. Values outside that couple can means leverage, partial buying/selling or holding. Trade signal can be created by humans after a market analysis or by statistical algorithms.

2.13 Market index

A market index is a simulated investment portfolio that reflects a section of the financial market. Different typologies of market indexes exists: some indexes have a price that is the weighted average of its constituents price, others have a return that is the weighted average of its constituents return. Different market indices are used by investors to track market changes. The prices of the underlying holdings are used to calculate the index value. There are different weighting methods for indices, here we are going only to name some: market-cap weighting, revenue-weighting, float-weighting, and fundamental-weighting. Famous indexes are the Dow Jones Industrial Average (DJIA), S&P 500, Nasdaq Composite Index, Eurostoxx 50, Nikkei 225. Investors cannot invest directly in an index, so these portfolios are used mainly as benchmarks.

2.14 Exchange Traded Funds and Index Futures

An exchange traded fund is a type of asset that have a price correlated with the value of an index, sector, commodity, or other asset, but which can be purchased or sold on a stock exchange the same way a regular stock can.

A **futures contract** is a legally binding agreement to buy or sell a certain asset at a defined price at a future date. Futures contract are traded on futures exchanges.

When a futures contract is purchased, the buyer assumes the responsibility to purchase and receive the underlying asset when the contract expires. The seller of a futures contract assumes responsibility for providing and delivering the underlying asset at the contract's expiration date. **Index futures** are contracts that allow a trader to purchase or sell an asset with the same price of an index today and have it resolved at a later date. Index futures are used by traders to speculate on an index's price direction.

Chapter 3

Deep learning methods

This chapter introduces machine learning architectures or layers that are used in this work, specifically Long Short Term Memory Networks (LSTM) Temporal Convolutional Networks (TCN) and Transformers. These are all architectures usually used for sequential data. Finally, we define types of errors in researching a machine learning model.

Machine learning has become key to important applications in science, technology and commerce. The focus of machine learning is on the problem of prediction: given a sample of training examples $(x_1, y_1), ..., (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}^p$ we learn a predictor $h_n : \mathbb{R}^d \to \mathbb{R}^p$ that is used to predict the label y of a new point x, unseen in training.

The prediction h_n is commonly chosen from some function class H, such as neural networks with a certain architecture, using empirical risk minimization (ERM) and its variants. We can see neural network as function class because neural networks model functions and by changing their weight we parametrize over a function space. In ERM, the predictor is taken to be a function $h \in H$ that minimizes the empirical (or training) risk:

$$\frac{1}{n} \sum_{i=1}^{n} l(h(x_i), y_i)$$

where l is a loss function, such as the squared loss $l(y', y) = (y' - y)^2$ for regression or 0-1 loss $l(y', y) = 1_{\{y' \neq y\}}$ for classification.

The goal of machine learning is to find h_n that performs well on new data, unseen in

training. To study performance on new data, knows as generalization, we typically assure the training example are sampled randomly from a probability distribution P over $\mathbb{R}^d \times \mathbb{R}$ and evaluate h_n on a new test example (x, y) drawn independently from P. The challenge arises from the mismatch between the goals of minimizing the empirical risk, the explicit goal of ERM algorithms i.e. optimization, and minimizing the true (or test) risk:

$$\mathbb{E}_{(x,y)\sim P}[\ell(h(x), y)]$$

the goal of machine learning.

3.1 Sequence modeling task

Before defining network architectures, we define the nature of the sequence modeling task. Suppose that we are given an input sequence $x_1, ..., x_N$ and wish to predict some corresponding output $y_1, ..., y_N$ at each time. The key constraint is that to predict the output y_t for some time t, we are constrained to only use those inputs that have previously observed: $x_1, ..., x_t$. Formally, a sequence modeling network is any function $f : X^t \to Y^t$ that produces the mapping:

$$\hat{y}_1, ..., \hat{y}_N = f(x_1, ..., x_N)$$

if it satisfies the causal constraint that y_t depends only on $x_1, ..., x_t$ and not on any future inputs $x_{t+1}, ..., x_N$. The goal of learning, in the sequence modelling setting, is to find a network f that minimizes some expected loss between the actual outputs and the predictions $L(y_1, ..., y_N, f(x_1, ..., x_N))$, where the sequences and outputs are drawn according to some distribution. This formalism covers many contexts such as auto-regressive prediction, where we try to predict some signal given its past, by setting the target output to be simply the input shifted by one time step. This formalism does not, however, directly capture domains such as machine translation, on sequence-to-sequence prediction in general, since in these cases the entire input sequence including future states can be needed to predict the output.

3.2 Recurrent neural network

The recurrent neural network (RNN) architecture was introduce by Rumelhart et al. [29] in 1986. RNNs operate on sequence of data by applying an identical function f to every element of the sequence to produce a sequence of state vectors $H = [h_1, ..., h_p]$. The function f takes as input both the output of the function at the previous point in the sequence, and the current element of the input sequence

$X = [x_1, ..., x_p]$. An RNN can be unfolded to represent it as a traditional feedforward neural network with no recurrence. A generic RNN is shown in figure 3.1.

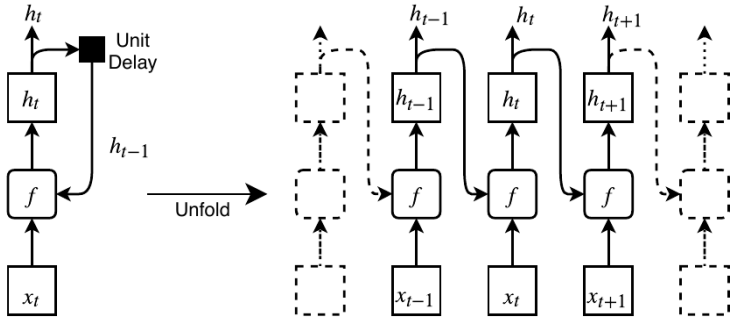


Figure 3.1: Generic recurrent neural network

At each point t in the sequence h_t is taken as the output. h_t can be projected to the required output dimension for example with a dense layer.

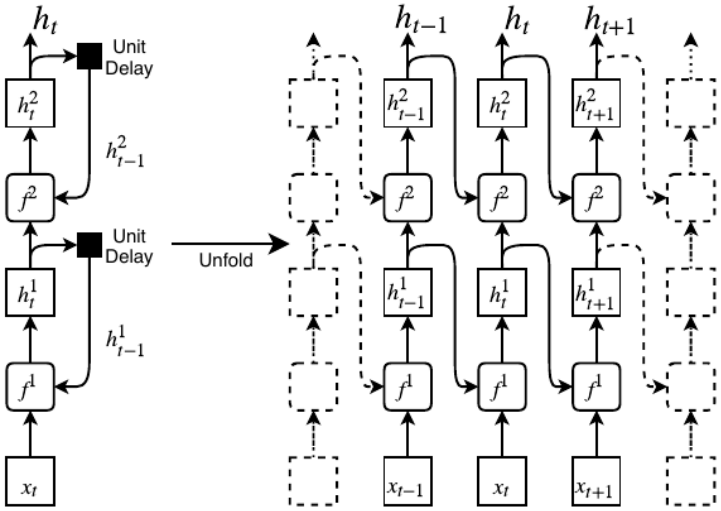


Figure 3.2: Two-layer recurrent neural network

RNNs can be extended to multiple layers, with a two-layer RNN illustrated in figure 3.2. Each layer implements its own function, with layer n implementing function f^n and producing output h^n. For layers after the first layer, instead of taking a x vector as input they take the state vector from the previous layer. There can be an arbitrary number of layers. A multi-layer RNN can in fact be thought as

a generalization of a single-layer, where multiple sub-function are included in the function f.

3.2.1 Long Short Term Memory Network

Recurrent networks are often used in sequence processing tasks because of their ability to capture temporal relationships. Long Short Term Memory (LSTM) is a modified version of recurrent neural networks proposed by Hochreiter and Schmidhuber [30]. Moreover, Recurrent Neural Networks are prone to vanishing and exploding gradients problems [31], which can be addressed through the use of long short term memory cell. An LSTM cell is able to selectively add and remove elements from the state vector as it passes through the cell. LSTM cells have been demonstrated to outperform simpler functions in a variety of tasks when employed as RNN function [32] [33] [34]. The LSTM cell extends the standard RNN model by separating the cell output c_t from the cell state h_t, shown in figure 3.3 and its multi-layer configuration shown in figure 3.4. The LSTM cell is described by the following equations:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3.1}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{3.2}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{3.3}$$
$$\hat{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{3.4}$$
$$c_t = f_t \circ c_{t-1} + i_t + \hat{c}_t \tag{3.5}$$
$$h_t = o_t \circ \tanh(c_t) \tag{3.6}$$

where:

- $\circ$ denotes the element-wise multiplication:
- the input vector is $x_t \in \mathbb{R}^n$
- the cell output vector is $h_t \in \mathbb{R}^d$ (d is the hidden dimension of the model);
- the cell state vector is $c_t \in \mathbb{R}^d$;
- the forget gate vector is $f_t \in \mathbb{R}^d$;
- the input gate vector is $i_t \in \mathbb{R}^d$
- the output gate vector is $o_t \in \mathbb{R}^d$
- the cell candidate vector is $\hat{c}_t \in \mathbb{R}$;

- the learned cell state weight matrices are $U \in \mathbb{R}^{d \times N}$;
- the learned input weight matrices are $W \in \mathbb{R}^{d \times d}$
- the learned bias vectors $b \in \mathbb{R}$; and
- $\sigma(y) = \frac{1}{1+e^{-y}}$ represents the sigmoid function

At first glance equations 3.1 to 3.6 may come across difficult, but they are in fact highly intuitive when studied alongside figure 3.3. There are three primary sub-sections within the LSTM cell, discussed in the three following paragraphs. Note that the use of two weight matrices (W and U) in the LSTM equations is equivalent to concatenating x and h and using a single weight matrix - keeping consistency with figure 3.3.

First, data is removed from the cell state (c) as it flows through the LSTM cell. The sigmoid function that produces f_t outputs values between 0 and 1. When these are multiplied with c_{t-1} (the previous cell state) some elements will be removed or reduced. The sigmoid layer that produces f_t is referred to as the *forget gate*, as it causes some data to be removed from the cell state depending on the previous cell output (h_{t-1}) and the current input (x_t).

Next, information is added to the cell state. The tanh layer produces a new candidate cell state ($\hat{c}_t$), and the input gate layer produces i_t in a similar fashion to f_t. The vector i_t decides which elements of the new candidate cell state vector are added to the actual cell state vector by reducing the magnitude of elements in $\hat{c}_t$ based on the previous cell output (h_{t-1}) and the current input (x_t). After $\hat{c}_t$ has been multiplied by i_t to selectively reduce elements it is added to the cell state. The sigmoid layer that produces i_t is referred to as the input gate, as it decides what data is added to the cell state as it passes through the cell.

Finally, a candidate output vector is produced by applying tanh element-wise to the cell state. The output gate layer produces o_t and this is multiplied with the candidate output vector to selectively remove elements, producing the final cell output h_t. The sigmoid layer that produces o_t is referred to as the output gate, as it decides which elements of the candidate output are passed as the final output.

Together, these gating mechanism allow the LSTM cell to selectively add and remove elements from the state as it passes through the cell, and allow the cell to selectively produce an output based on the current cell state. This generally allows the LSTM cell to retain data from many timestep in the past, making it a superior choice to simpler functions when working with long sequences of data, though in some cases the LSTM cell can discard data from early in the sequence.

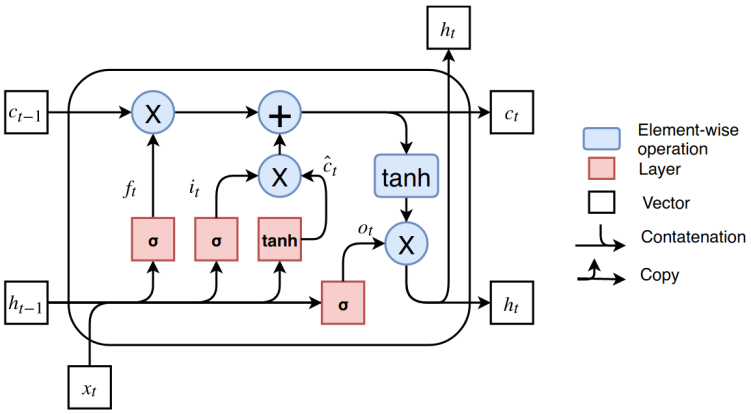


Figure 3.3: Long short-term memory cell.

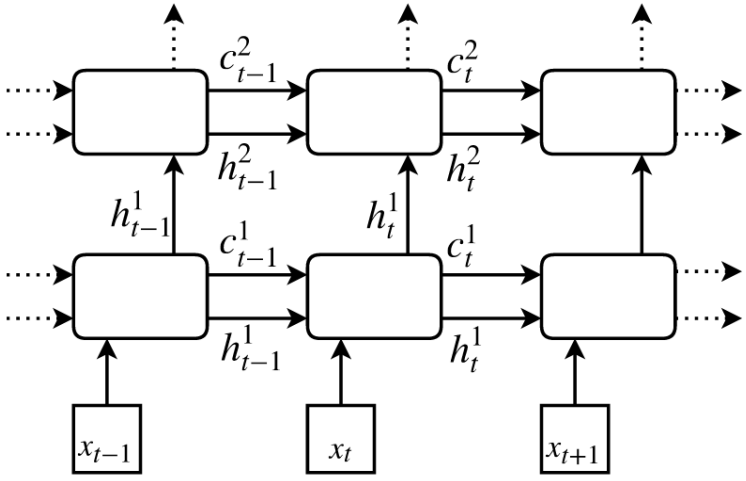


Figure 3.4: Multi-layer long short-term memory cell configuration.

3.3 Temporal Convolutional Networks

Temporal Convolutional Networks were introduced in 2016 by Lea et al. [35]. TCNs are one-dimensional causal dilated Convolutional Neural Network. In this section we define TCNs that will satisfy the following properties: (1) computations are performed layer-wise, meaning every time step is updated simultaneously, instead of updating sequentially per-sample (2) convolutions are computed across time, (3) prediction at each step are a function of a fixed-length period of time, which is referred to as the receptive field (4) the convolutions in the architecture are causal, meaning that there in no information leakage from future to past (5) the architecture can take a sequence of any length and map it to an output sequence of any length, just as with an RNN. We will show how TCNs can build very long effective history sizes (i.e. the ability for the network to look very far into the past to make a prediction) using a combination of very deep networks, augmented with residual layers, and dilated convolutions. Figure 3.5 illustrates an example of Temporal Convolutional Network. TCN tends to be superior in many task to LSTM and also Bidirectional LSTM [36].

3.3.1 Temporal convolution

Let $X_t \in \mathbb{R}^{F_0}$ be the input feature vector of length F_0 for time step t for $1 \leq t \leq T$. A TCN consist of L layers denoted by $E^{(l)} \in \mathbb{R}^{F_l \times T_l}$ for $1 \leq l \leq L$ where F_l is the number of convolutional filters in the $l-$th layer and T_l is the number of corresponding time steps. We define the collection of filters in each layer as $W = \{W^{(i)}\}_{i=1}^{F_l}$ for $W^{(i)} \in \mathbb{R}^d \times F_{l-1}$ with a corresponding bias vector $b \in R^{F_l}$. Given the signal from the previous layer, $E^{(l-1)}$, we compute activations $E^{(l)}$ with

$$E^{(l)} = f(W * E^{(l-1)} + b)$$

where f is the activation function and $*$ is the convolution operator.

3.3.2 Dilated causal convolution

When dealing with Convolutional Neural Networks (CNNs), *causal* means that each output y_t of the network is exclusively a function of previous input timestep $x_0, ..., x_t$, while *dilated* means that each hidden layer is given a *dilatation factor d* which controls which timestep from the previous layer the convolution is actually applied over. A simple causal convolution is only able to look back at history with size linear in the depth of the network. This makes it challenging to apply the before mentioned causal convolution on sequence tasks, especially those requiring longer history. The solution here is to employ dilated convolutions that enable an exponentially large receptive field. Given a sequence $x = x_0, ..., x_n \in \mathbb{R}$ and a filter

$f : 0, ..., k-1 \rightarrow \mathbb{R}$, the *dilated causal convolution* operation $x *_d f$ on the t-th element of x is defined as follows:

$$(x *_d f)(t) = \sum_{t=0}^{k-1} f(i) \cdot x_{t-d \cdot i}$$

where d is the dilation factor, k is the filter size, and $s - d \cdot i$ accounts for the direction of the past. Note how $t - d \cdot i$ enforces the *causal* part of the convolution, as it can only point to sequence elements which are in the past. Dilation is equalent to introducing a fixed step between every two adjacent filter step. When $d = 1$, a dilated convolution reduces to a regular convolution. The dilatation factor d usually increases the further we get into the network, often taking the values of increasing powers of two (i.e. $d = 2^i$ on the i-th hidden layer of the network). This ensures two things. First, that the receptive field of the network grows exponentially with the number of layers (whereas in a undilated CNN this would grow linearly) granting a longer history. Second, that the output y_t at time t is effectively a function of a contiguous number of steps in the input. A network with m layers has a receptive field of size $2^{m-1}k$. Therefore there are two ways to increase the receptive filed of the TCN, choosing a larger filter size k and increasing the dilation factor d.

3.3.3 Residual connections

A residual block (He et al. 2016) [37] contains a connection leading out to a series of transformations F, whose outputs are added to the input x of the block.

$$o = Activation(x + F(x))$$

This effectively allows layers to learn modifications to the identity mapping rather than the entire transformation, which has repeatedly been shown to benefit very deep networks.

Since a TCN's receptive filed depends on the network depth n as well as filters size k and dilation factor d, stabilization of deeper and larger TCNs becomes important. For example, in a case where the prediction could depend on a history of size 2^{12} and a high-dimensional input sequenced, a network of up to 12 layers could be needed. Each layer, more specifically, consists of multiple filters for feature extraction.

However, whereas in a standard Residual Network (ResNet) the input is added directly to the output of the residual function, in TCN (and other Convolutional

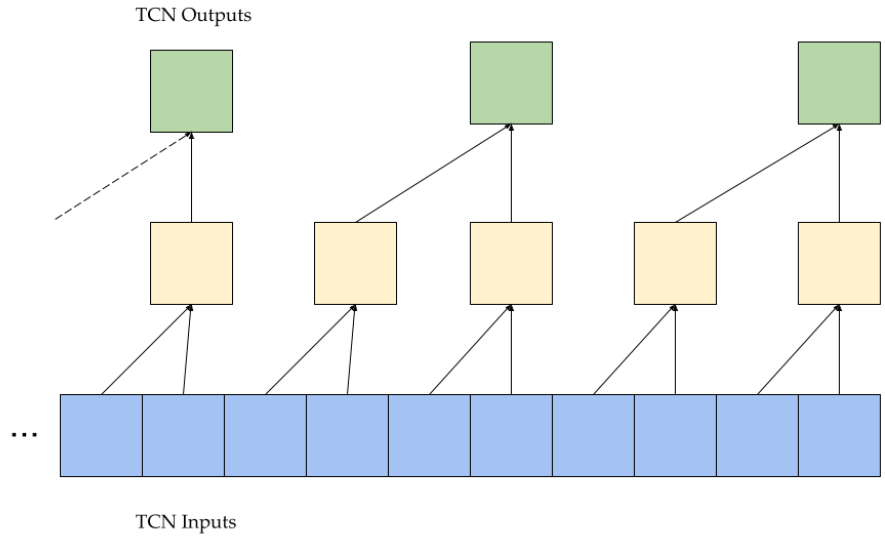


Figure 3.5: Temporal Convolutional Network with output size 3 and two hidden layers each of dilatation factor 2

Networks in general) the input and output could have different widths. To account for discrepant input-output widths, we use an additional 1×1 convolution to ensure that element-wise addition $\bigoplus$ receives tensors of the same shape.

3.4 Transformers

The Transformer neural model architecture, shown in figure 3.6, was introduced by Vaswani et al. [1] in 2017 and at the time was the state of art in neural machine translation. Neural machine translation is the task where a machine translate from a language to another.
Compared to another typical architecture to deal with time series data, like the Long Short Term Memory network (LSTM) [30], Transformers do not compresses information as time goes by. Such compression can weak long-distance relation patterns to some extent and may fail to highlight important information from historical data. Using **attention layers** (see section 3.4.2) it's possible to overcome this problem. Unlike previous sequence to sequence models, attention models do not process data sequentially but process all the sequence together. Attention layers

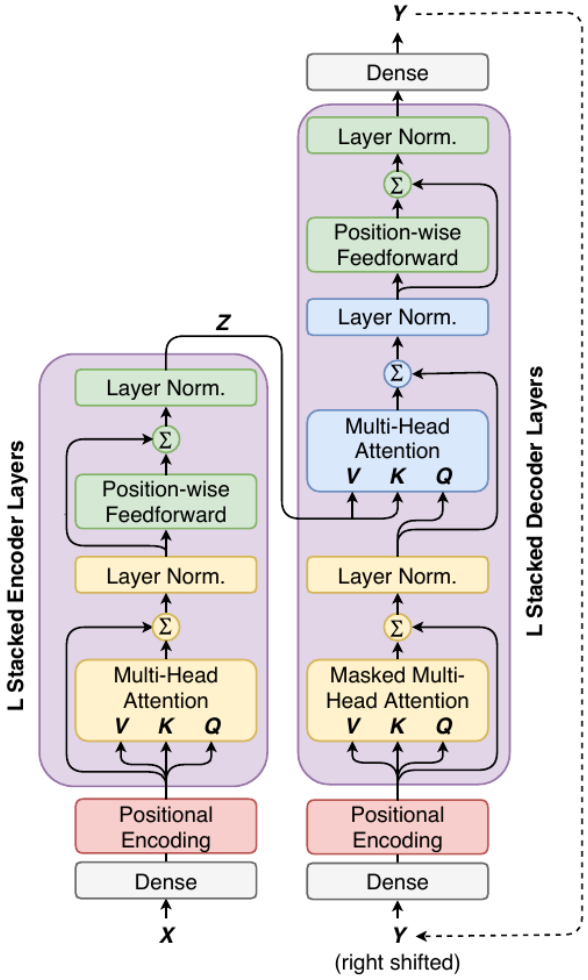


Figure 3.6: Transformer architecture [1]

learn temporal and spatial dependencies from the sequence. By processing all the sequence elements simultaneously, models that use attention are highly parallelizable and can take advantage of the computational power of GPU outclassing on performance LSTM.

The transformer architecture follows a similar sequence-to-sequence/encoder decoder architecture. The encoder transformers an input sequence $X = [x_1, ..., x_p]$ into a latent representation $Z = [z_1, ..., z_m]$, and the decoder transforms Z into an output sequence $Y = [y_1, ..., y_r]$.

The original Transformer was meant for language translation, a task where input data and output data have the same structure/typology, they are essentially phrases of a natural language. Our case is different, as in input we have different features derived from returns in the form of time series but output is a weight

vector. This is the reason why in the original transformer shown in figure 3.6 the latent representation of X is inserted in the middle of the decoder. The original transformer find matching between words of phrase X and phrase Y, the essence of machine translation task, in their respective latent space. But as we are not doing machine translation, but regression, we remove the decoder part and we use the latent representation of X namely Z directly to compute the final network output through a Dense layer. We call this reduced architecture **Regression Transformer**. A representation of the regression Transformer can be see in figure 3.7.

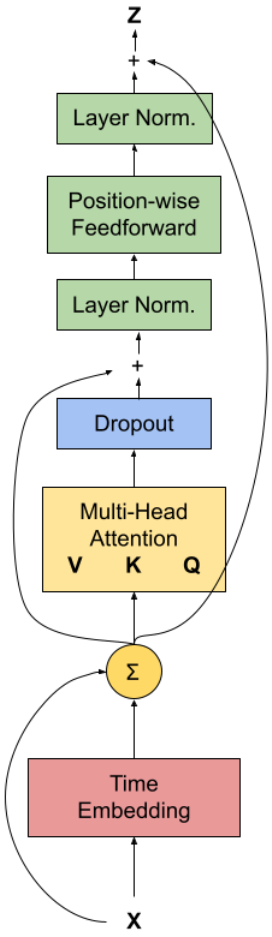


Figure 3.7: Regression Transformer architecture

3.4.1 Time embedding

For Attention to work, an encoding of time need to be attached to the input features. In the original NLP model, a collection of superimposed sinusoidal functions were added to each input embedding. We need a different representation as our inputs are scalar values and not distinct words/tokens. Deep Neural Networks can learn linear and periodic components on their own, during training using a Time2Vec layer [38]. Time2Vec is a learnable and complementary, model-agnostic representation of time. It decompose each input feature to a linear component and as many periodic (sinusoidal) components it's necessary. By defining the decomposition as a function, we can make the weights learnable through back propagation. For a given scalar notion of time τ, Time2Vec of τ, denoted as $t2v(\tau)$, is a vector of size $k+1$ defined as follows:

$$t2v(\tau)[i] = \begin{cases} \omega_i\tau + \phi_i, & if\ i = 0. \\ F(\omega_i\tau + \phi_i), & if\ 1 \leq i \leq k. \end{cases}$$

where $t2v(\tau)[i]$ is the i-th element of $t2v(\tau)$. F is a periodic activation function, and ω_is and ϕ_is are learnable parameters. Given the prevalence of vector representation for different tasks, a vector representation for time makes it easily consumable by different architectures. We choose F to be the sine function in our experiment. The period of $\sin(\omega_i\tau + \phi)$ is $\frac{2\pi}{\omega_i}$, i.e. it has the same value for τ and $\tau + \frac{2\pi}{\omega_i}$. Therefore, a since function helps capture period behavious without the need for feature engineering. For istance, a since function $sin(\omega\tau + \phi)$ with $\omega = \frac{2\phi}{7}$ repeats every 7 days (assuming τ indicates days) and can be potentially used to model weekly patterns. The linear term represent the progression of time and can be used for capturing non-periodic patterns in the input that depend on time.

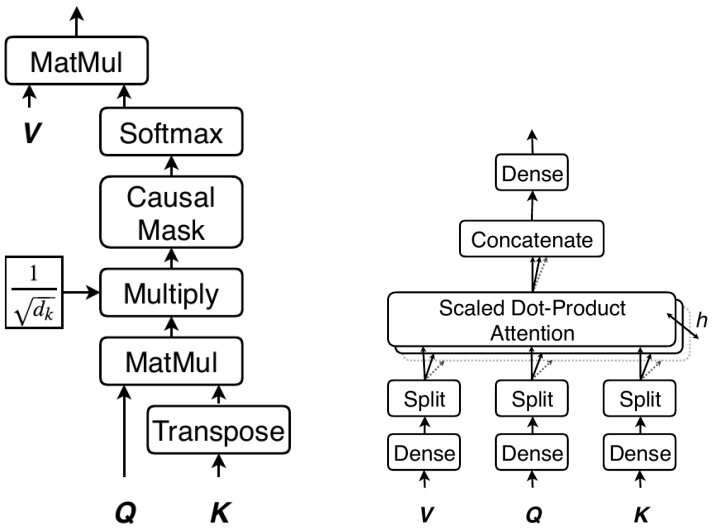
3.4.2 Multi-head attention

The primary innovation of the Transformer architecture is multi-head attention. Generic attention and dot-product attention will now be described as there are prerequisite to describing multi-head attention.

An attention function can be described as mapping a query and a set of key-value pairs to an output where the query, keys, values and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

In the *scaled dot-product attention* the input consists of queries and keys of dimension d_k, and values of dimension d_v. The dot products of the query with all keys is computed. This dot product is divided by $\sqrt{d_k}$ to prevent the dot product from becoming large when d_k is large as this may cause the softmax gradient to become very small and affect gradient descent training, and then a softmax function is applied to obtain the weight on the values. In practice, the attention function is computed on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. The matrix of output is computed as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



(a) Scaled dot product attention

(b) Multi-head attention

Figure 3.8: Attention mechanism used in transformer architectures.

Vasvani [1] also proposed multiheading shown in figure 3.8 (b). It applies a separate dense layer to each of the values, queries and keys. The dense layer is applied with learned weight $W \in \mathbb{R}^{d \times d}$ and a learned bias vector $b \in \mathbb{R}^d$, where d is the hidden dimension of the model, a hyperparameter. The outputs of the dense layers are then split along the last axis into h sets, or heads. As a result the key, query and value dimension is reduced by a factor of h to $\frac{d}{h}$. Scaled dot-product attention in then run independently on each set. The results are concatenated and put through

a final dense layer to produce the output of the attention function. The dense layer function on the output has learned weights $W \in \mathbb{R}^{d \times d}$ and learned bias vector $b \in \mathbb{R}^{d \times d}$.

The dense layer combined with the split allows the multi-head attention to pick out information from different subspaces in the input and direct these to different attention heads. This is in contrast to a single head which must average all subspaces.

3.4.3 Residuals and Normalization

Residual connections [37] are applied around each sub-layer. That is, the output of each sub-layer is given by $X' = X + subLayer(X)$ where $subLayer(X)$ is the original output of the sub-layer. The outputs are then normalized by applying layer normalization [39].

3.5 Adam optimizer

Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. It was introduced by P. Kingma in 2014 [40]. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. The name Adam is derived from adaptive moment estimation. Adam is designed to combine the advantages of two other methods: AdaGrad, which works well with sparse gradients, and RMSProp, which works well in online and non-stationary settings. For additional details and the presudocode of the method see the original paper [40].

3.6 Errors and model capacity

In supervised machine learning we can divide the prediction error in 3 components [41]: bias, variance and irreducible error:

$$E_m = E_b + E_v + E_i$$

Irreducible error is the error caused by the fact we do not have available all possible instances of the type of training data we use. For example if we want to build a classificator for bird species it would become a perfect classifier if we had all the possible pictures of birds to train with, but unfortunately is impossible. An

example in finance would be that we do not have all possible market outcomes that can ever be possible in a stock exchange in the past and in the future. Generating them would be impossible because we do not know which future company would be in the market and how investors would react to it. This result in the fact that there is always a situation where the network is unprepared to deal with. Irreducible error can be very small, but it always exists.

Model capacity is a measure of the ability of the model to learn complex functions from the data. Model capacity is the **dimension of the class** H to which the predictor h_n, described in the introduction of this chapter, belongs. Alternatively, but similarly, it can be defined as the number of parameters needed to define a function in the class H. Capacity is a trade-off, a medium point is preferable. Too high capacity leads to fitting the training data optimally, to have a small empirical risk but poor generalization on other datasets. This is the case of overfitting. Too low capacity and our model could not be able to learn the target function, to have a large empirical risk. This is the case of underfitting. The control of the function class capacity may be explicit, via the choice of H, for example by picking the neural network architecture, or it may be implicit, for example using regularization, for example internal parameters constraints. When a suitable balance is achieved, the performance of h_n, on the training data is said to generalize to the population P.

For example in k-nearest neighbors algorithm, the higher we set k in the model, the lower is the capacity, the boundaries between classes are smoother and generalize better, until it reaches a point where boundaries are too simple and too many inputs are misclassified. Alternatively, with $k = 1$ every point in the input is correctly classified in its class, but these boundaries fits very well the training dataset and only a slight variation of input characteristics in the validation datasets could lead to a high classification error. The model memorized the training dataset. Capacity is a characteristic of the architecture, not of the instanced model. The instanced model is characterized by its weight values and it can be overfitted or underfitted depending on how much it has been trained. Figure 3.9 show the relation between error rate, model capacity, bias, variance and irreducible error.

Given n training/validation datasets, n different models can be trained. Bias is the performance difference between model i and the best model between the n. Variance is the difference between i and the performance value average of all the n models.

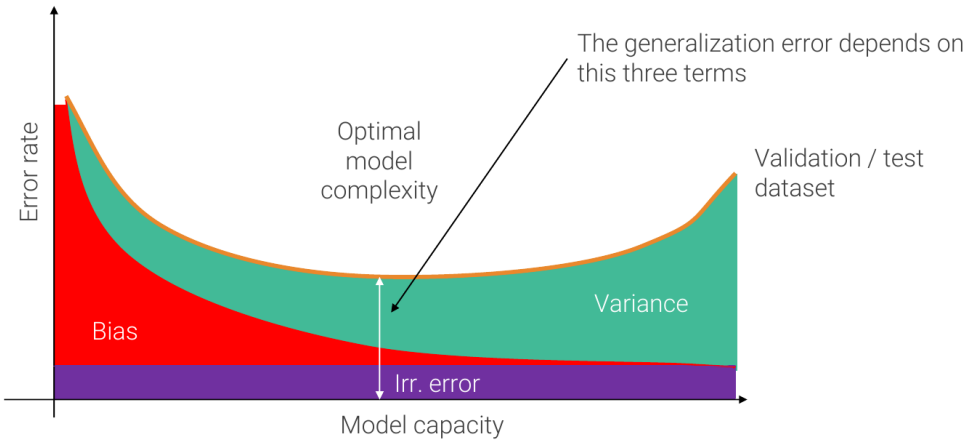


Figure 3.9: Bias-Variance-Irreducible error during training [42]

Another way to reduce variance, other than choosing a better model architecture or using more data, is regularization. Regularization is any modification we make to a training algorithm that is intended to reduce its generalization (test) error but not its training error. Examples of regularization are weight constraints, early stopping, dropout, etc...

Chapter 4

Methodology

This chapter is structured as follows. Section 4.1 introduced our research objective and what we want to attain. Section 4.2 described the input data our model is trained and where is coming from. Section 4.3 introduced our method of research for a suitable model. Sections 4.4 and 4.5 describe two different models that differ by the loss functions used. Section 4.6 deals with reproducibility of results and how we achieved it. In section 4.7 we described a problem with the model output and how we solved it. Finally, section 4.8 explains how we find the best value for some general hyperparameters.

4.1 Problem formulation and scope

We search for a neural network that takes as input an array $x \in R^{b \cdot s \cdot f}$ with dimensions $b \cdot s \cdot f$ where b is the batch size, s is the sequence length and f the number of features. The model has to give as output an array $y \in \mathbb{R}^n$ with dimension n equal to the number of assets in the portfolio. In our specific case $n = 4$ (see below for the considered assets).

We compare the allocation method with 4 benchmarks: a baseline algorithm described next, equally weighted allocation, inverse volatility and risk parity. The last 3 methods are defined in Chapter 2.
The **baseline allocation algorithm** is based on Sharpe ratio (see section 2.7). Each asset weight is first ordered by decreasing Sharpe ratio, then to each assets a weight is assigned in the order 40%, 30%, 20%, 10%. This is the current algorithm used by Salzenberg AI, the company we are collaborating with, and the percentages values have been chosen from their empirical experience with financial markets.

Our objective allocation method has to have a minimum single asset allocation of 5%. An upper bound of 60% for a single asset was decided based on empirical experience on financial markets. These bounds will be implemented subsequently in the model.

4.2 Input data

Our input data are different features obtained from daily returns of an investing strategy. The investing strategy is provided by a company with we are collaborating and is based on four market indices as it trades futures linked to them. First we explain this four market indices, then some more details about the strategy and finally how which features we obtained from daily returns. In this work we mainly focus on four market indices The market indices are:

- Nasdaq 100 made up of stocks issued by 100 of the most capitalized companies of the Nasdaq American stock market. It is capitalization weighted.
- Standard and Poor's 500 made up of 500 largest American companies traded on stock exchanges in the United States. It is capitalization weighted.
- EURO STOXX 50 made up of fifty of the largest stocks traded in the Eurozone. It is capitalization weighted.
- Nikkei 225 is a price-weighted index of 225 large companies in Japan, traded on the Tokyo Stock Exchange.

Indices are not directly tradable but commons proxy of them are ETFs and Futures (see section 2.14), even if differences are presents. The model is not trained on returns of the indices, ETFs or Futures, but on *strategy returns* over them. We create strategy returns for each of the 4 assets by multiplying the returns of a linked future with the values of a trading signal. The trading signal is provided by Salzenberg AI, a fintech company which provides trading signals for a managed investment fund. Their strategy is both long and short (see section 2.1), it trades intraday without overnight exposure. This means all investments are sold before market close at day t and reinvested at day $t + 1$, so the actual return is only computed between opening and closing price of the same day. We are not authorized to disclose additional details of the strategy more than this. But this is not relevant for this thesis since our objective is to find an optimal allocation *given* some trading signals. For our research intentions and objectives, training on the original index returns or on the proprietary strategy is the same. The baseline algorithm taken into account in this thesis is the current allocation algorithm used at Salzenberg

and is the goal of our collaboration with them to find an allocation method that would perform better that it.

From returns different operators can be applied to obtain different features which in turn can be combined to form different feature sets. The features used in this work are:

- returns
- 15 days moving average of returns
- 15 days moving standard deviation of returns
- Cumulative returns, see section 2.1
- Relative Strength Index (RSI), see section 2.10

Trading volumes were tested as feature, as they are useful for predicting future volatility as said in section 2.11, but were constantly excluded from the feature selection algorithm. One of the reasons may be that the trade volume used were from an ETF linked to the indexes, that not always is correlated to index volumes.

A feature sets generator, a Python function, has been created and generates all possible combination of features of different length. n features can generate $2^n - 1$ not-empty features sets. In our case $n = 5$ so the feature list of sets has length 31. The feature selection algorithm is run to select which feature sets results in the best performance depending on the metric chosen: Sharpe ratio or cumulative return.

Since features have a different magnitude, they have been scaled all in the interval 0-1. Feature scaling is also important to regularize training. If features are not scaled gradient descent is harder as different steps size have to be computed for each feature.

4.3 Model research method

Figure 4.1 describes how the three architectures in analysis, namely LSTM, TCN,

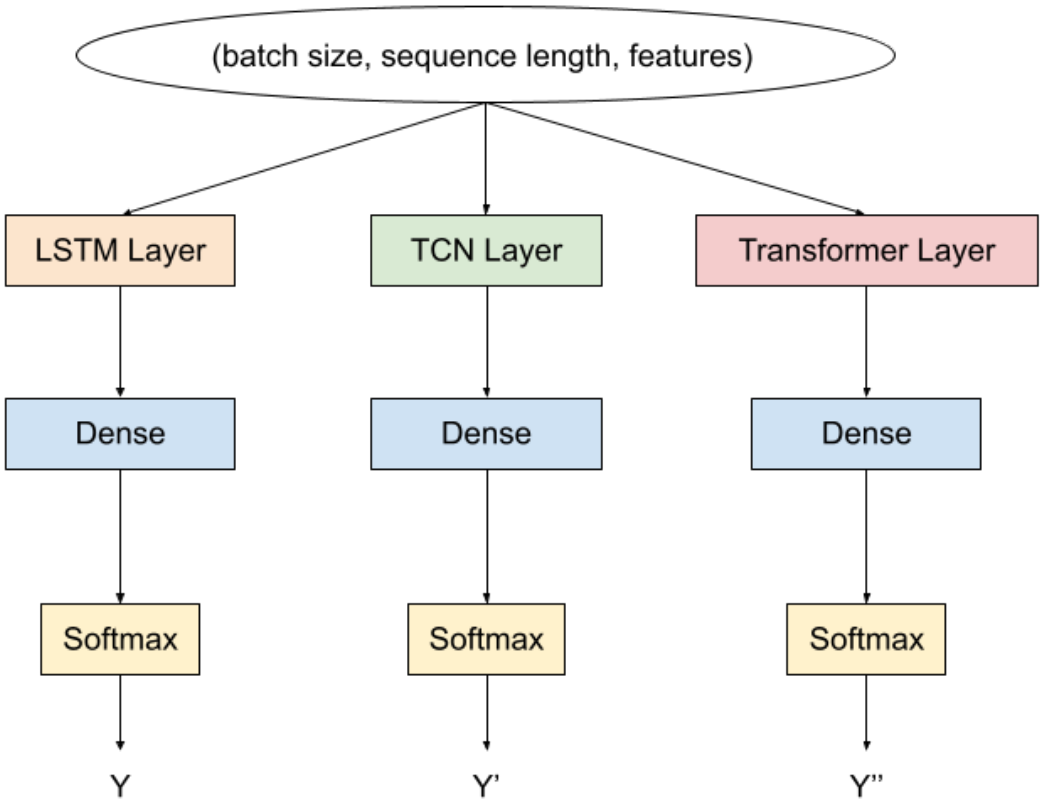


Figure 4.1: Diagrams of architectures in analysis

and Transformers, were used. We can think of the three architectures as a building block of a more common architecture for generating allocations. Indeed one can notice in the figure before mentioned a certain degree of similarity. We can abstract and see the three architectures nothing more than another layer of an architecture. Now, we define an architecture were:

- The input is a vector $x \in \mathbb{R}^3$ with dimension $(b \times s \times f)$ where b is the batch size, s is the sequence length and f the number of features
- the first layer is a parameter, than can be either an LSTM, TCN or Transformer layer.
- Since each architecture have a different internal dimension their output dimension is different so we add now a Dense layer with output dimension four. This add another level of expressivity to our architecture.
- It is not guaranteed that the output values of the Dense layer are a weight

vector that sum up to one. Therefore, as Zhang [6] does, in all the tested models the last layer is always a *softmax* layer. A softmax layer is a layer that implements the softmax function:

$$softmax(\bar{x}) = \frac{e^{\bar{x}}}{\sum_{i=0}^{n} e^{\bar{x}_i}}$$

This is often used to to normalize the output of a network to a probability distribution or as in our case to normalize values as allocation weights, to force them sum up to one.

Entering more in details into the architectures, LSTM is as described in section 3.2.1 with specific parameters the number of units and the number of LSTM layers. TCNs is as described in section 3.3 with temporal dilated convolution and residual connection. The Transformer architecture used is the regression transformer described in section 3.4 with residual connections, layer normalization and dropout. Table 4.1 list all hyperparameters of the architectures in consideration.

Model / method	Parameter	Description
Common to all methods	b	Training bath size
	e	Training epochs
	lr	Learning rate
	s	sequence lenght
LSTM	d	internal/hidden dimension, units
	l	numer of layers
TCN	d	internal/hidden dimension, units
	l	numer of layers
	k	kernel size
	sk	residual connections (Yes/No)
Transformer	h	number of attention heads
	hd	internal/hidden dimension, units
	o	output dimension
	l	number of layers

Table 4.1: Architecture hyperparameters

When searching for a model in machine learning, the problem is essentially about finding the proper input data, architecture and hyperparameters. Each of these has

an impact on the others. This means that changing one parameter or component may result in another parameter or component no longer being the best option. The safest choice would be doing a grid search, trying all the possibilities to find the best combination that maximize performance. However, this is often not possible for time and performance reasons. In this work we followed an order of operation that first select the most critical parameters and then fine tuning less impactful one, in an iterating manner, as follows:

1. We started only with daily returns as input features, because is a small feature set that allows us to experiment quickly and is the base feature from which all others features are generated. Technically every network could generate all the other features from returns, so this input feature is an adequate starting point.

2. As hyperparameters we started with batch size 100, learning rate 10^{-2} and sequence length 256 and 5 epochs.

3. We started using the TCN to find out which features were the most useful and had greater impact.

4. Then hyperparameters like learning rate, batch size and epochs were corrected without changing inputs features and architecture.

5. Then with the new hyperparameters, we try different architectures.

6. Lastly, fine tuning hyperparameters is again performed, especially the one of the architecture.

We applied several techniques of **holdout validation**. Initially the training period was 2013-2018 with 2019-2020 as validation set. But in 2020 the market was in a very special phase, caused by the COVID-19 pandemic and the March 2020 stock markets crash. This caused that networks outperforms in the validation set because they are not trained for a very bear market and the rapid recovery that happened afterward. The model was suffering from high variance (see Chapter 3). 3-fold validation is then used and specifically:

- Training on 2013-2018 and validation on 2019-2020
- Training on 2015-2020 and validation on 2013-2014
- Training on external periods 2013-2015, 2018-2020 and validation on the middle period 2016-2017

In this way we select models that are overall satisfactory in the most preponderant market phases. In each period the model is re-initialized twice and trained twice. The results are averaged. The same is done for the results of all periods for having an indicative, general across-dataset metric. This is done for avoiding considering wrong models where the model learn the wrong general function, but that is instead good for a specific case. For example the allocation $1/n$ is extremely easy to learn, good in many situations (see [17] [18] [19]) but not in all.

All the models have been implemented in open-source machine learning framework TensorFlow [43]. Development has been carried out using Jupyter Notebooks [44], as they allow a faster visualization of results.

4.4 Sharpe ratio maximizing model

As in *Zhang et al. (2020)* [6], an inverted **Sharpe ratio maximizing function** has been initially used. The function calculates the Sharpe ratio of the portfolio with the weights produced by the network. The code of the loss function can be found on the listing 4.1 .

The Sharpe ratio sign is then inverted to minimize it. Weights produced by the network are saved in a temporary array in a FIFO data buffer manner to be used for next Sharpe ratio computations. The Sharpe computation is made inside a context of automatic differentiation thought GradientTape API. After this, gradients are explicitly computed and applied.

A normalization layer had to be implemented, to force the model to output weights that are between the 5% and 60% chosen bound, as specified in the section 4.1. This is because maximizing Sharpe ratio does not guarantee the allocation weights will be contained in that range. This normalization layer has been implemented directly as a Keras layer, instead of normalizing the weights after they have been generated from the network. This is to enforce the loss to be computed on correct and operational weights, to constrain the model to learn to produce an allocation we can use in our scenario. The code of the normalization layer can be seen in the listings 4.2 and is inspired by the following formula:

$$(t_{max} - t_{min}) - \frac{m - r_{min}}{r_{max} - r_{min}} + t_{min}$$

```
@tf.function
def sharpe(output, past_data, past_weights, today_data):
    # expected return / std dev of porfolio
    # get returns from today_data
    # multiply with network choosed weight
    weighted_returns = tf.multiply(output,today_data)
    # get comulated return of last period
    past_returns = past_data
    weighted_past_returns = past_returns*past_weights
    cumulative = tf.math.reduce_prod(
        (1+weighted_past_returns)
        ,axis=0)
    # mutiply today data with cumulative from past
    total = (cumulative * (weighted_returns+1))-1
    # std_dev of last period
    weighted_past_returns =
        tf.squeeze(weighted_past_returns)
    std_dev = tf.concat(
        [weighted_past_returns,weighted_returns]
        ,axis=0)
    std_dev = tf.math.reduce_std(std_dev,axis=0)
    # ratio
    sharpe = total/std_dev
    return tf.math.reduce_mean(sharpe)
```

Listing 4.1: Python function to compute the portfolio sharpe given new assets weights from neural network

Where:

- m is the value to be scaled
- r_{min} minimum in the range of values
- r_{max} maximum in the range of values
- t_{min} minimum in the desired target scaling
- t_{max} maximum in the desired target scaling

Listing 4.2: Keras layer to force the weights to be between 5% and 60%

```
min_bound = 0.05
max_bound = 0.6
def normalization_layer(x):
    minV = tf.repeat(tf.expand_dims(
        keras.layers.Minimum()(tf.unstack(x,4,axis=-1))
        ,axis=1),4,axis=-1)
    maxV =  tf.repeat(tf.expand_dims(
        keras.layers.Maximum()(
            tf.unstack(x,4,axis=-1))
        ,axis=1),4,axis=-1)
    return (max_bound-min_bound)
                *((x-minV)/(maxV-minV))+min_bound
```

4.5 Target allocation matching model

Next, a different type of model was tested. Instead of maximising the Sharpe ratio we use a network which still generates weights but the loss function is the mean squared error (MSE) between these generated weights and some optimal weights. Formally, given the vector $w = [w_1, w_2, w_3, w_4]$ generated by the network and the vector $y = [y_1, y_2, y_3, y_3]$ of the target allocation, we define the MSE loss in this context as:

$$MSE(w, y) = \frac{1}{4}\sum_{i=1}^{4}(y_i - w_i)^2$$

In this example we used four as number of assets but this definition can be easily extended to any number of assets.

The optimal weights are generated for the training set through an algorithm with knowledge of future returns. This algorithm can make use of Markowitz optimization (see chapter 2 and PyPortfolioOpt in the Appendix) or other basic algorithms like the following one:

Algorithm 1 Caption

Order assets comulated returns in a window in decreasing order.
Assign respectively 40%, 30%, 20%, 10% to the first, second, third and forth asset.

4.6 Reproducibility

An experiment is **reproducible** when it can be executed multiple times and obtain the same result. In the context of machine learning this is not easy to achieve. Framework like Keras have several random initialization, especially of network layer kernels. The reason is to increase the likelihood of starting exploring the parameter space in a gradient slope and increase the possibility of convergence. But to obtain reproducibility, it is then suggested to initialize the weights to the same value and if not possible to initialize their random generator to the same initial seed.

Reproducibility is an important aspect of this work especially when in the phase where different features sets were tested. Once a feature set was discovered to be upstanding with respect to the others, it was run on a fresh copy of the model to verify the results. But by default TensorFlow resets the kernels and other parameters every time to random values. It was then necessary to fix the random seed to a fixed value in each notebook. Moreover, the kernel of each layer has been initialized to a unit value for the first layers of each model, and to a normal distribution but with unit seed for the fully connected last layer. Initializing the last layer to a unit value was not guaranteeing convergence. This effectively improved the reproducibility of the results.

4.7 Fixed allocation anomaly

The weights generated by the network, with both loss function methods, initially had the problem of being fixed even as the inputs changed. This can be seen in the right part of plot *a* in figure 4.2. Allocation weights change during training but they remain the last computed by the backpropagation algorithm in the last iteration of the training algorithm. During training they change only because the internal weights of the network change, but in validation when the internal weights are not allowed to change, also the output remain fixed.

The reason of this phenomenon is that the network only found the best bias value to increase the overall performance, both in terms of Sharpe ratio or of return. The input weights are null, that means the network does not use the input features to compute the output.

Although the performance on the validation dataset was indeed good on the first days, because it fitted the market phase of the last period of training set, it degraded

over time. This was the case the validation set was temporally right after the training set, in the different cases the fixed allocation resulted in a bad performance from the very beginning. In general, this was a case when the training algorithm was stopped in a local minimum, the network found out how to minimize the target loss but did not find a relationship between input and optimal outputs.

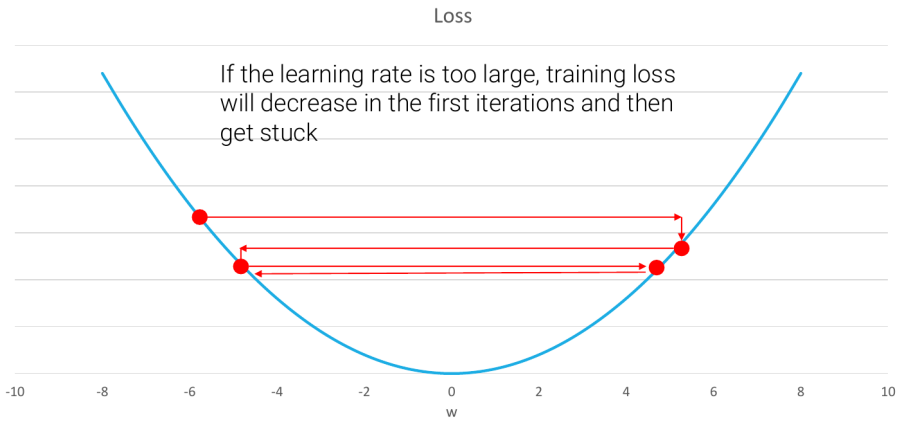


Figure 4.3: Excessive learning rate effects [42]

Most of the time, this situation was solved by reducing the learning rate. **Learning rate** is a fundamental hyperparameter. A too large learning rate, while it can make convergence faster, increases the likelihood of the training getting trapped in an oscillation like shown in figure 4.3. Instead, a too small learning rate makes training too slow and the algorithm could convergence very late or never converge if not enough training time is provided.
The initial learning rate of 10^{-2} was changed to 10^{-3} with radical changes in the output quality. In this case the gradient was persisting in an oscillatory state in a in the parameters space, finding a sub-optimal solution changing bias value and ignoring input values. By lowering the learning rate and increasing the number of epochs, the gradient would slowly direct the training to a state where inputs were effectively used and a more optimal portfolio allocation was found.

In cases were lowering the learning rate was not sufficient, we resorted to force **constraints on the biases** of each layer by using the constraint `max_norm(0)` provided by TensorFlow. This forces the network to use the inputs for providing the output and effectively finding a relation between them. In the figure 4.2 we see

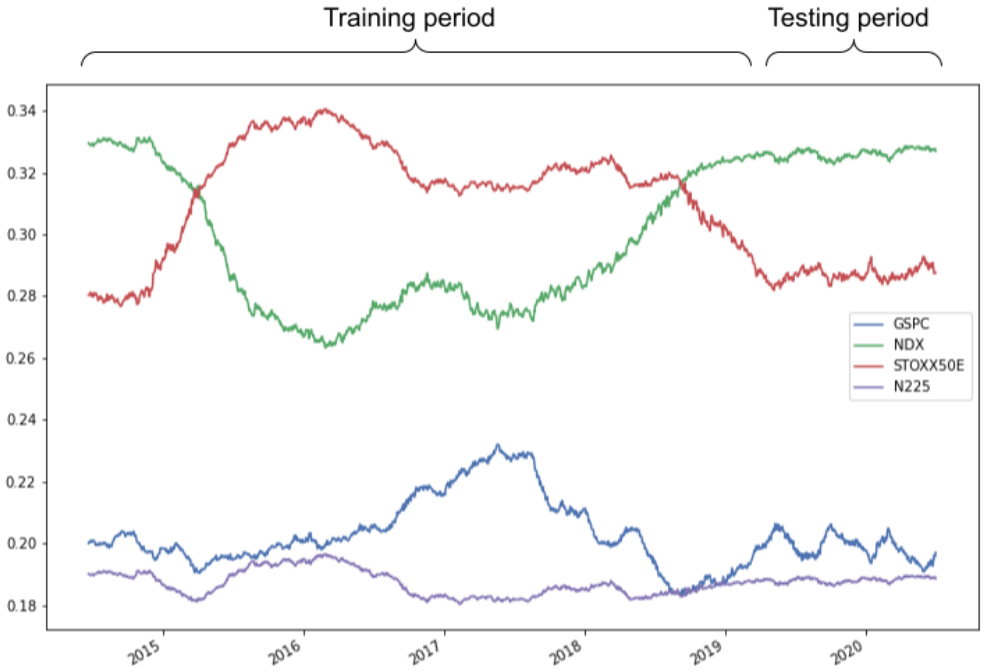
the difference in computed allocation weight by a fixed situation and a dynamic one.

4.8 Remaining hyperparameters

Batch size was initially set to a value of 100. With time we noticed that a smaller batch size, for example of value 5, was better. The reason is the following. The value of the loss function is averaged always across the batch size. A too large batch size makes the loss value too general and uniform across the training. The result is that some micro-variations in the market of the duration of few days are lost and the network is not learning how to react and take advantage of them. On the opposite situation, having a batch size too small, for example only one sample, is disadvantageous. This is because the network would pay too attention in the inherent noise that is inside a market and would be unable to highlight weekly trend. The gradient would change direction contentiously from sample to sample, unable to find a common minimum in the parameter space. It must be considered that even if the time frame is not negligible this is not a high trading frequency application as many others in this field and only daily data is available. This makes the dataset considerably small with respect to other machine learning applications. It is therefore essential to have have a reduced batch size.

Experiments with different windows size have been carried out. A smaller window size allows the model to react faster to sudden market phase change while a longer window allows the model to understand long-term factors and implication in the market and invest in more stable assets.

(a) Not optimal allocation, is fixed in testing period



(b) optimal allocation, is dynamic in both train and test dataset

Figure 4.2: Fixed and not-fixed allocations

Chapter 5

Results and discussion

This chapter presents our research results. The results are obtained following methodology described in Chapter 4, especially Section 4.3. A comparison between the two loss functions, the one maximizing Sharpe ratio and the one that match a optimal allocation is presented. After this, we show the results of the feature selection algorithm. With the loss function and the feature set fixed we compared the three different architectures in analysis: LSTM, TCN, and Transformer and we show here the results. It is also explained which parameters of the architectures are chosen. Once we define architecture and parameters, we show an important result, that the return, Sharpe ratio e standard deviation of the generated allocation is influence by the target allocation respective parameters. This allows a control on the Mean-Variance trade-off for the network user. Finally, chosen a target allocation, the performance of the selected model compared to the aforementioned benchmarks in the three validation periods described in Chapter 4 is extensively described.

5.1 Choice of loss function

We ran the same network architecture with the same parameters on the same input dataset with the two different loss function, the one maximizing the Sharpe ratio described in Section 4.4 and the one trying to match a target allocation described in Section 4.5. On average, the Sharpe ratio maximizing method has a training from 3 to 15 times slower than the allocation matching one.
We were not able to reproduce Zhang [6] results. We used the same hyperparameters, i.e. 50 LSTM units, with a small learning rate of 10^2 and even by forcing the bias to zero, the result is always a fixed not optimal allocation anomaly as described in Section 4.7. In the validation period July-December 2014, it results in a cumulative return of 13.7%, the same as the baseline algorithm, while the method using

allocation matching scores a 15% cumulative return. The latter is matching an allocation generated by the algorithm 1 at page 59 using a window of 256 days.

Moreover, using this second type of loss function, it is not necessary to insert a normalization layer, since the weights constraints are implicitly already present in the target allocation. This also improves the computational speed of the network.

However, it must be taken in account that the Sharpe ratio maximizing method takes risk into consideration, while the other method tries to match an allocation that it has been computed only considering the maximal return. It comes natural to think that the network is learning to perform two different tasks, so the previous comparison is not so applicable. Indeed, the Sharpe ratio maximizing method has, on the validation period aforementioned, a mean Sharpe ratio value of 3.53 while the second, that instead indirectly maximizes returns, has a Sharpe Ratio of only 3.29.

We repeated the comparison a second time by targeting, with the second loss function, an allocation generated in such a way that it would maximize the Sharpe ratio. Our hypothesis is that the model is learning to produce an allocation with the same characteristics as the one it was trained with. The situation is similar for what is happening for the weights constraints. If the model is trained on an allocation always between 5% and 60%, it will also remain in this boundary when validation data is given. The network will do this because in training it has been penalized for going outside of this boundary. The same holds for the risk control, if a Sharpe ratio maximal allocation is produced, e.g. though PyportfolioOpt, and then a model is trained targeting it, the resulting generated allocation will have be the one of a Sharpe maximizing portfolio.

By repeating the experiment this time targeting a different allocation as described in the previous paragraph, the Sharpe ratio of the allocation generated is 3.50. More on these results can be seen in Section 5.5.1. This is an important result because it proves our hypothesis and the flexibility of the second method, that other than being faster, allows the investors to have some sort of risk control by choosing the risk level of the target allocation with which the model is trained. From now on the only loss function that will be used will be the second one, the allocation targeting method. We will target an allocation generated for maximizing the return.

5.2 Feature selection

The following table summarizes the results of feature selection. A green cell indicates when the corresponding feature is used, while a red cell when a feature is not used. In the table we show only the top 6 results from the 31 possible feature combinations. Results are ordered by decreasing mean cumulative return, of the portfolio following computed allocation, in the 3 different evaluation periods described in Chapter 4.

Portfolio cumulative mean return	Daily Returns	Daily returns 15 days standard deviation	Daily returns 15 days mean	Cumulative returns	RSI
20.2%					
19.9%					
19.8%					
19.7%					
19.6%					
19.6%					
19.3%					

Analysing the table, we can note a predominance of the cumulative returns in all the features sets. On the top seven feature set, cumulative return is always included. This can be interpreted as for the network is easier to compute if a component in the time frame is increasing in value overall of decreasing. It is sufficient to look at the last value of the cumulative return series that will contain the cumulative return of the time frame in analysis. Also as said in the work of [13] cumulative returns do not have the short term noise from the daily return. The second most important feature are raw daily returns. All the other features are optional, they could improve or worsen the results, depending on the analysed period.

5.3 Architecture selection

Using daily returns and cumulative returns in a window of 125 trading days, we train and evaluate the 3 architectures in analysis with the methods described in Chapter 4. In table 5.1 we explicit the performance in each period. The hyperparameters of the architectures have been chosen to have a similar number of internal weights, namely of the order of 36 thousand. The temporal convolutional network performs better than the other two architectures with a cumulative return higher up to an increase of 4.7%.

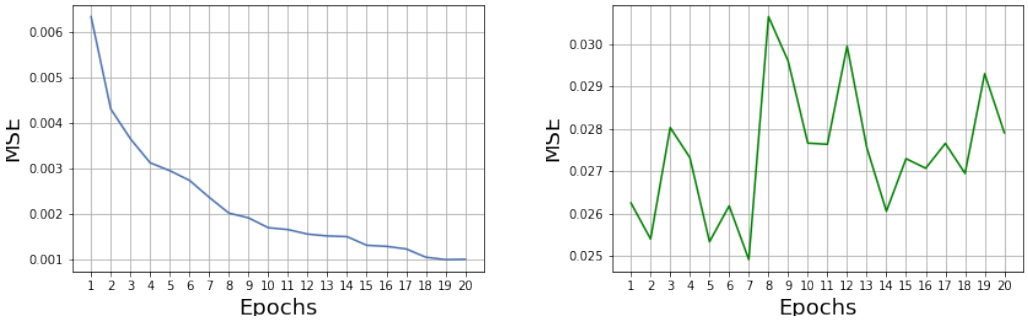
	June 2014-December 2014	June 2016-December 2017	June 2019-June 2020
LSTM	12.1%	13.0%	26.9%
Transformer	12.8%	14.2%	24.3%
TCN	**15.5%**	**14.2%**	**29.0%**

Table 5.1: Architecture comparison: cumulative return in 3 different time frames

Here the training lasted 5 epochs. If trained for longer periods, for example 10 epochs, the Transformer performs better than the TCN but only in the 2014 period. It is out of the scope of this work to find a reason of the performance differences between the architecture, still we show in figure 5.1 the target allocation, computed with algorithm 1 at page 59, and the output allocation of each model in the 2014 period. This shows the different approaches that every architecture has.
LSTM and Transformer perform poorly in the period shown because they do not invest enough in the EUROSTOXX 50, that is the 1st or 2nd best asset in the period.

Regularization techniques have been tried, by both inserting dropout in the TCN layer or a batch regularization layer after it. The results showed how these techniques bring little improvements if not worsen the results, therefore it have been decided to not use them in the final model.

5.4 Hyperparameter selection



(a) Training loss (b) Validation loss

Figure 5.2: Model convergence: epochs and training/validation loss

The TCN model converge as shown in the fig 5.2 (a), the training loss decrease constantly during training. This is the minimization of the Empirical Risk discussed in chapter 3. Instead, TCN has problems of overfitting over six epochs, as shown in fig 5.2 (b). Therefore we limited the number of epochs to five. This figure is the result of validating the model in the 2017 period. The reason why we have to limit ourselves to a so little epochs number can be explained by the fact we also have a little batch size (we use batches of dimension five). These two quantities are often related, as a small batch size make the network more sensible to small scale training data variations so less epochs are required to model these.
Models with a small window size, like 10 trading days, perform better in 2020 in a very volatile market phase, while models with a window size of 225 trading days perform better in the previous period 2013-2019 when the market was more stable. The window size of 125 trading days, around 6-7 full months, have shown to deliver the best results so far as it is a compromise between fast market change reaction and long term investment in reliable assets.

Regarding the other hyperparameters, a grid search using the TCN architecture have been carried out, with these ranges: number of epochs = [5, 10], TCN filters = [8, 16, 32], convolution kernel size = [1, 2, 3], residual connection = enabled / disabled.
The results do not underline a trend, and it is impossible to say that a specific choice will for sure improve results. Still we kept the hyperparameters of the best results, namely 5 epochs, 32 filters, kernel size 3 and no residual connection.

5.5 Selected model results

A model has been selected following decisions described in the sections before.

5.5.1 Generated allocation parametrization

We show that there is a certain degree of correlation between parameters of target allocation, i.e. Cumulative return, Sharpe ratio, and Volatility and the same parameters in the generated allocation, allowing risk control management. Table 5.2 shows the results of thirteen runs in the 2014 period of the selected model on different target allocation, the left part of the table, and the resulting performance parameters of the generated allocation (see the right part of the table).

The correlation between these quantities is shown in Table 5.3. Unfortunately the outcome is that only target and generated standard deviation are correlated (0.7)

Target return	Target Sharpe	Target std		Generated return	Generated sharpe	Generated std
30.1%	4.35	0.40		31.2%	4.44	0.41
44.9%	4.45	0.59		25.5%	3.02	0.49
39.2%	4.56	0.50		26.7%	3.37	0.46
43.1%	4.59	0.55		29.8%	3.88	0.45
35.4%	4.66	0.44		31.8%	4.36	0.42
43.6%	4.69	0.54		38.9%	4.28	0.53
40.8%	4.69	0.51		31.1%	3.78	0.48
37.6%	4.71	0.46		27.3%	3.72	0.43
41.6%	4.74	0.51		33.0%	3.82	0.50
42.8%	4.79	0.52		39.5%	4.38	0.52
42.8%	5.77	0.43		28.3%	3.99	0.41
63.3%	6.93	0.53		32.0%	4.14	0.45
62.2%	7.00	0.52		29.0%	3.92	0.44

Table 5.2: Resulting performance parameters of generated allocation given parameters of target allocation in the 2014 period

while return and Sharpe ratio are not.

This fact can be explained that often the target allocation are too hard for the network to match, either a too high return, where the network is unable to find out which components will perform better in the future, or a too low volatility, where the network is unable to predicting which components will have a lower volatility in the future. It exist a certain threshold over with the network stop performing better and instead perform worse than before, this explain some of the negative correlations.

Instead, by relaxing the constraints and considering only a subset of the previous thirteen experiments we obtain far better results. We restrains the runs that have a cumulative return lower than 43% and a standard deviation higher than 0.44. We are left with six out of the original thirteen, that are still a significant sample and we again compute a correlation table shown in table 5.4.

We can observe as the correlation between target and generated standard deviation is very high, it is easier for the network to predict future volatility than future returns. Still also the the target cumulative return and the generated cumulative

	Target return	**Target Sharpe**	**Target std**	**Generated return**	**Generated Sharpe**	**Generated std**
Target return	1.00	0.89	0.52	-0.00	-0.08	0.11
Target Sharpe	0.89	1.00	0.08	-0.06	0.15	-0.24
Target std	0.52	0.08	1.00	0.08	-0.49	0.70
Generated return	-0.00	-0.06	0.08	1.00	0.72	0.60
Generated Sharpe	-0.08	0.15	-0.49	0.72	1.00	-0.12
Generated std	0.11	-0.24	0.70	0.60	-0.12	1.00

Table 5.3: Correlations between target and generated allocation parameters in the 2014 period

	Target return	**Target Sharpe**	**Target std**	**Generated return**	**Generated Sharpe**	**Generated std**
Target return	1.00	0.52	0.97	0.59	-0.02	0.98
Target Sharpe	0.52	1.00	0.29	0.78	0.62	0.55
Target std	0.97	0.29	1.00	0.44	-0.20	0.93
Generated return	0.59	0.78	0.44	1.00	0.79	0.71
Generated Sharpe	-0.02	0.62	-0.20	0.79	1.00	0.13
Generated std	0.98	0.55	0.93	0.71	0.13	1.00

Table 5.4: Correlations between target and generated allocation parameters on a restricted set of experiments

return have a correlation coefficient of 0.59 that is still significant. Ultimately, also the Sharpe ratio is correlated, as it is the ratio between the two previously mentioned quantities.

This indicates again that is possible to control the result allocation parameters and have risk control with the method researched and selected in this work.

Table 5.5 summarizes the selected model performance in comparison to the benchmarks: baseline algorithm, equally weighted, inverse volatility and risk parity benchmarks.

	June 2014 December 2014	June 2016 December 2017	June 2019 June 2020
Baseline algorithm	1.4%	0.9%	8.0%
Equally allocation	1.5%	1.6%	3.5%
Inverse volatility	2.3%	4.5%	1.6%
Risk Parity	4.3%	3.3%	0.6%

Table 5.5: Performance difference (cumulative return) between selected model and benchmarks methods

What follows are sections describing the performance both in terms of cumulative return and of Sharpe ratio in the validation periods described in Chapter 4.

5.5.2 2014 period

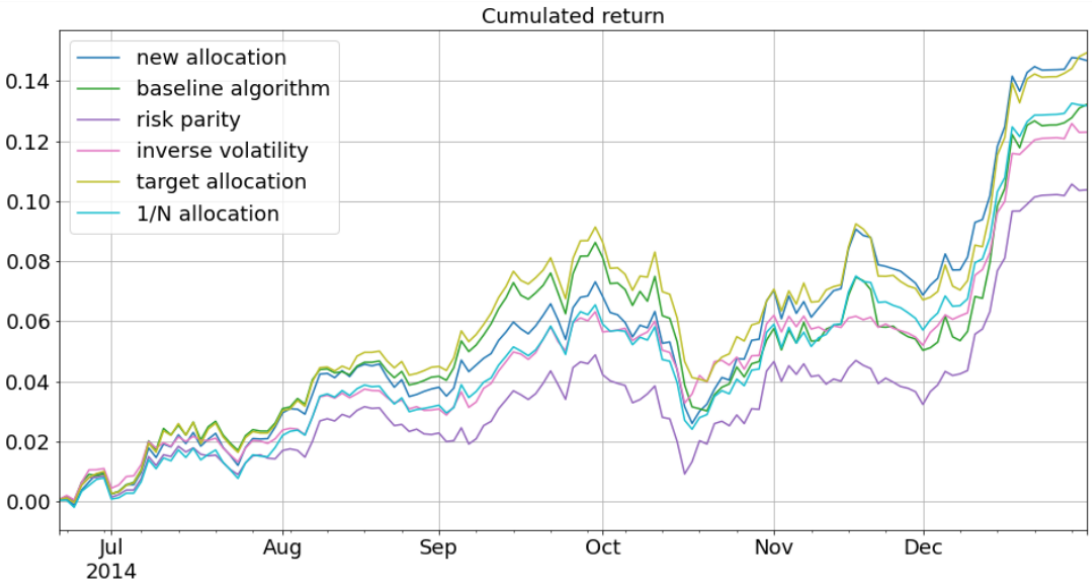


Figure 5.3: Selected model performance: Cumulative return, July-December 2014

2014 is the year were the network performs better among the analysed periods, because it matches perfectly the target allocation in the second half of the time frame. Figure 5.3 shows the cumulative return between July 2014 and December 2014. Our allocation method performs better than all the benchmarks in consideration.

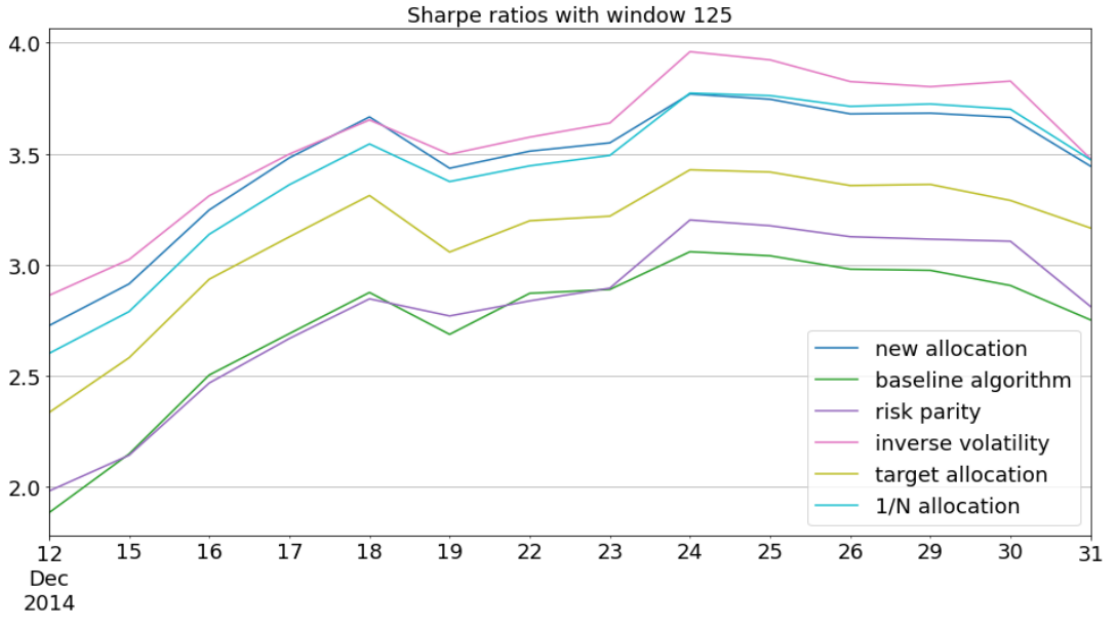


Figure 5.4: Selected model performance: Sharpe ratio, December 2014

In December 2014 period the Sharpe ratio is also considerably satisfactory, as seen in Figure 5.4, as it is the highest between the benchmarks until December the 23rd and then it decreases only surpassed by inverse volatility and equally weighted methods. The 2014 period is only 6 month long and 125 days are need to compute the Sharpe ratio, so as result we only have 31 days were the Sharpe ratio is computed, in December 2014.

5.5.3 2016-2017 period

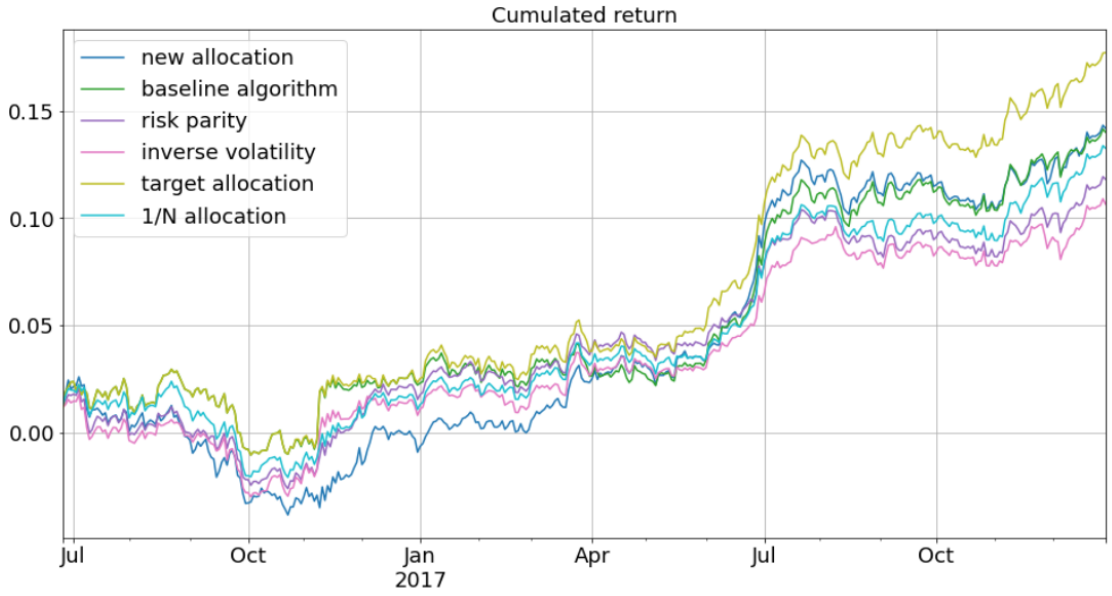


Figure 5.5: Selected model performance: Cumulative return, July 2016-November 2017

In the validation period July 2016 - November 2017 our method does not result in a higher cumulative return than the target allocation, like in the validation period July-December 2014, but it still outperforms all other benchmarks, as one can see in Figure 5.5.

Sharpe ratio is shown between January and December 2017 in Figure 5.6. Our method results in a Sharpe ratio in average with other benchmarks for most of the periods. It is under performing in the first months of January to March 2017 while it is superior between June and September 2017. However, it should be noted that also the target allocation does not perform better in term of Sharpe ratio with respect to the other benchmarks. This is because this allocation was created maximizing return and not Sharpe ratio.

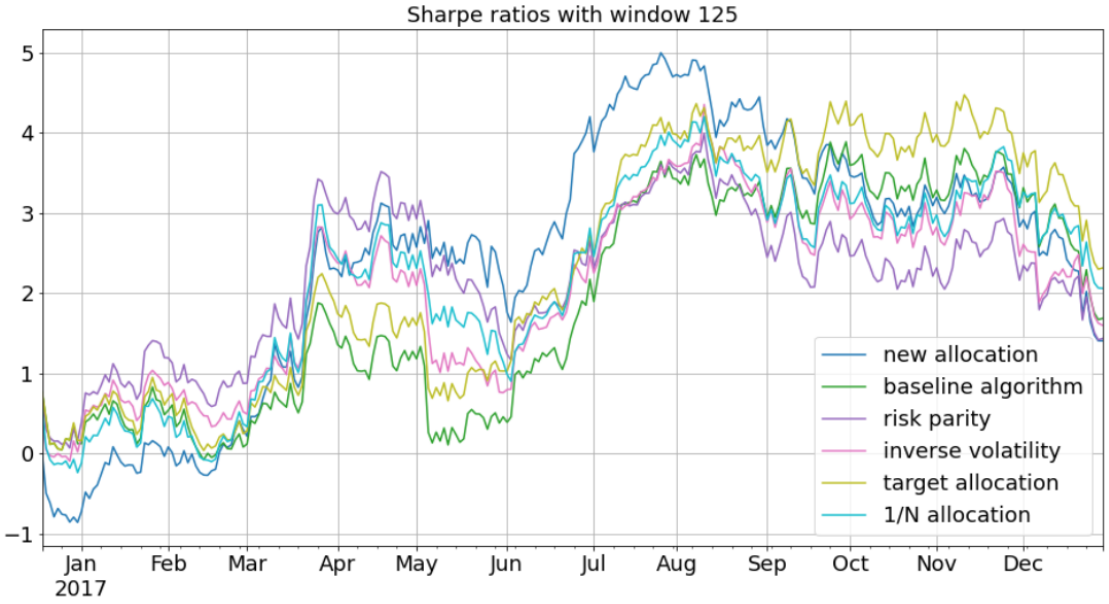


Figure 5.6: Selected model performance: Sharpe ratio, January-December 2017

5.5.4 2019-2020 period

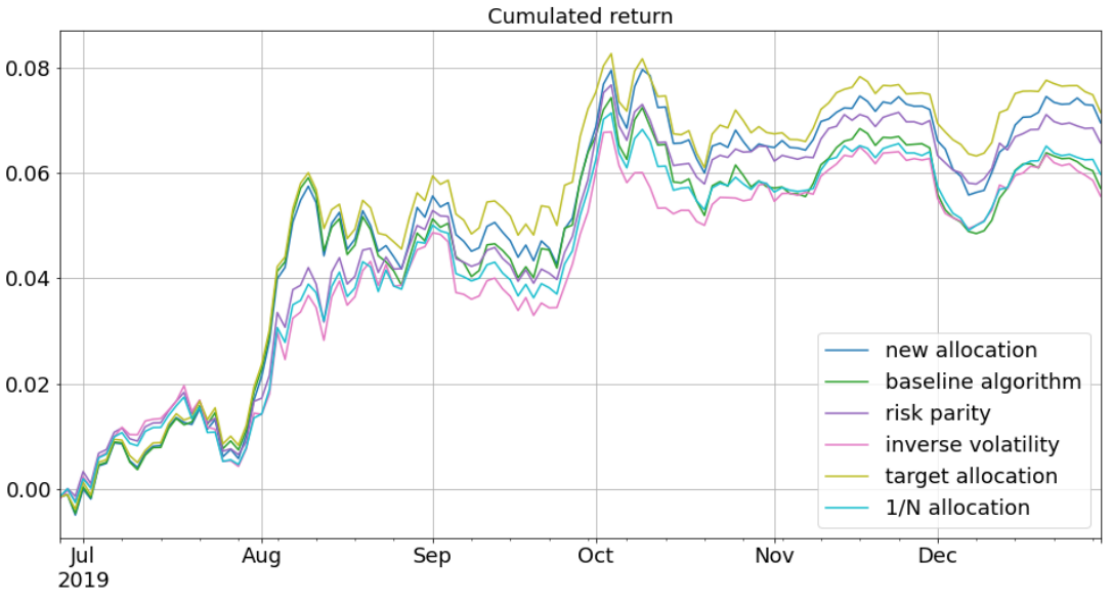


Figure 5.7: Selected model performance: Cumulative return, July 2019 - January 2020

In the validation period 2019-2020 our method has two different phases so the cumulative return plot has been spitted in two parts for an easier comprehension. Figure 5.7 shows the time frame July 2019 to January 2020 where the market was still growing and the benchmarks have all similar cumulative returns values. Still our model generates an allocation that in the end is superior to the baseline algorithm, the inverse volatility, and equally weighted allocations.

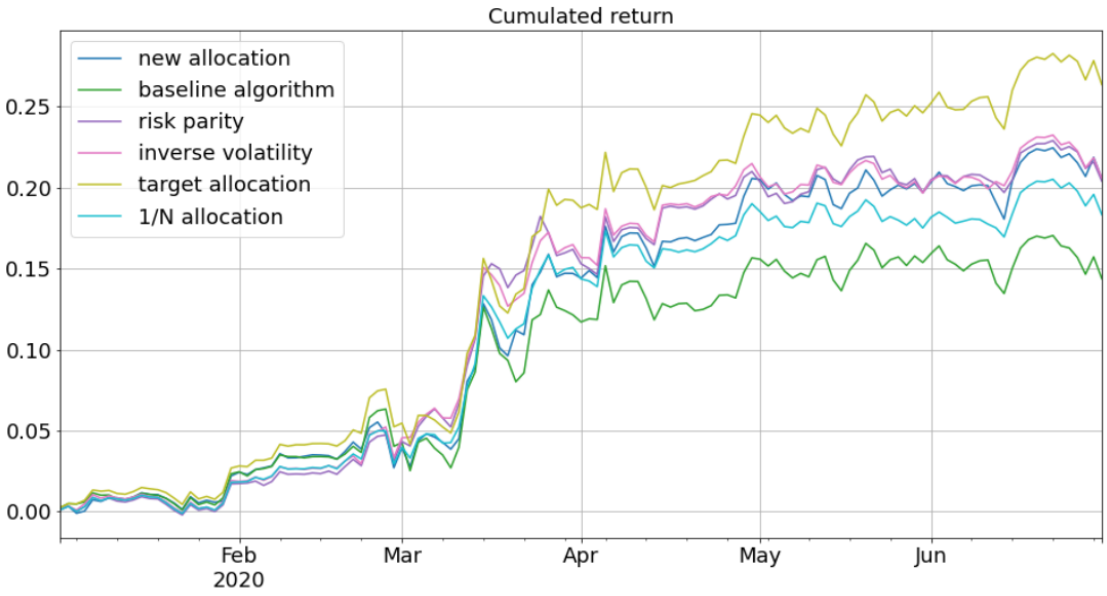


Figure 5.8: Selected model performance: Cumulative return, January-July 2020

Within the time frame from January to July 2020, the 2020 stock market crash is included , but the strategy we used went short in that period so a surge in the cumulative return can be seen in March 2020 in Figure 5.8. Before March 2020 all the figures perform similarly but after they diverge significantly with our allocation method being superior to both baseline algorithm and equally weighted allocation and consistent with inverse volatility and risk parity.

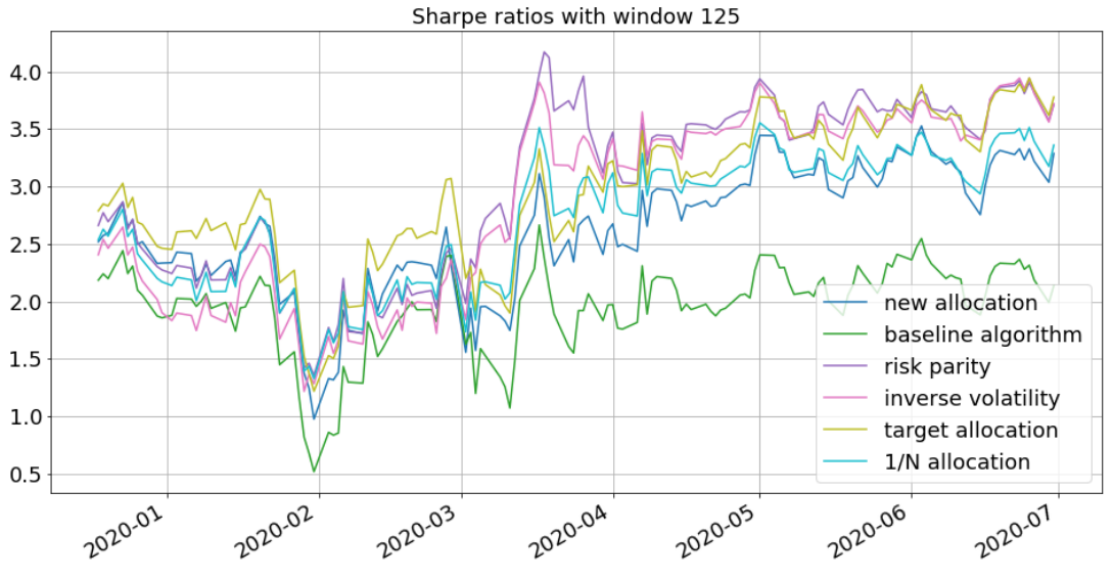
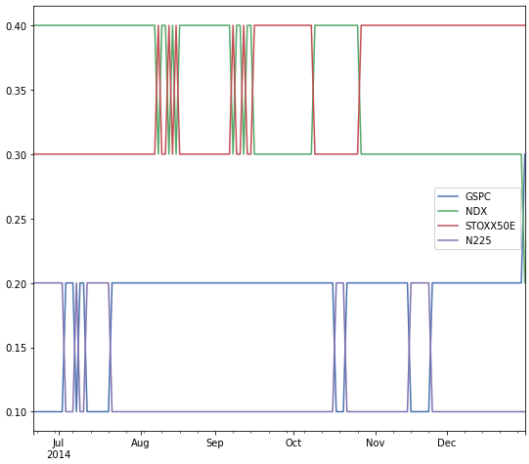
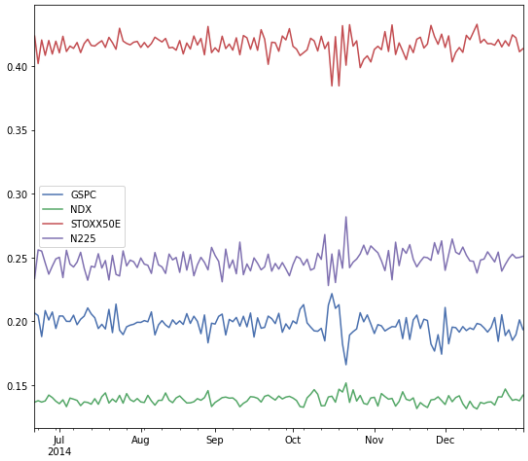


Figure 5.9: Selected model performance: Sharpe ratio, January-July 2020

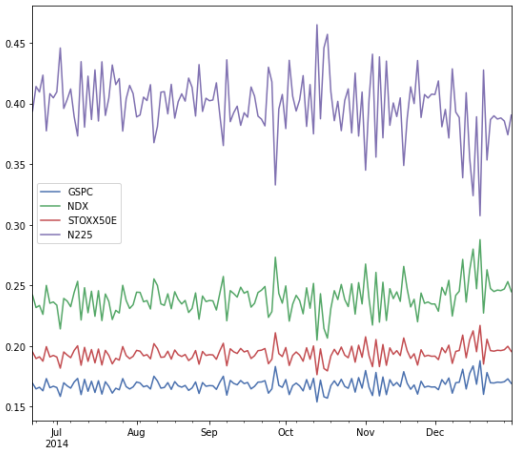
Sharpe ratio of the allocations in analysis for the period January to July 2020 can be seen in Figure 5.9. Here our allocation method performs poorly compared to the others and is inferior to most of them. Still it must be reminded that this allocation is not meant to have a high Sharpe ratio and it has a higher volatility compared to the other. We believe this explain this result.



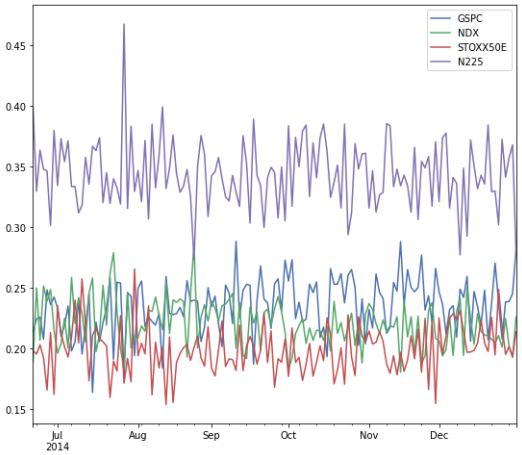
(a) Target optimal allocation



(b) Temporal Convolutional Network allocation



(c) Long Short Term Memory Network allocation



(d) Transformer allocation

Figure 5.1: Allocations generated from TCN, LSTM e Transformers

Chapter 6

Conclusions

This thesis researched a Machine Learning model that generates an optimal portfolio asset allocation starting from daily returns of these assets. The model was found to generate an allocation with a cumulative return always superior than the baseline algorithm and the equally weighted allocation. Compared to the other two benchmarks, inverse volatility and risk parity, the model performs similarly or better. Increase of cumulative return range from $1.6\% - 3.5\%$ annually in comparison with equally weighted to $1.6\% - 4.5\%$ with inverse volatility and $0.6\% - 3.3\%$ with risk parity. Different input features, all generated from daily returns, were tested and it is suggested to combine daily returns with cumulative returns. Different architectures were examined, namely Long Short Term Memory Network, Temporal Convolutions Networks and Transformers. TCNs was the architecture that guaranteed the best performance. Different techniques of hyperparameters optimization and regularization were applied to guarantee model convergence. Finally, the models work by training on matching a target allocation in a training dataset. This allocation can be generated with classic methods of portfolio optimization like Markowitz, targeting a maximum Sharpe ratio or maximum return. The model will then learn to generate an allocation with the same characteristics. This gives our model also a control of risk.

6.1 Future work

A first improvement would be having more data available, simply extending the period before 2013 could make the network better in predicting market phases.
The target allocation used had, on purpose, a low variability because it has been noticed models had difficulty matching an allocation that would change nearly every day. Still, better allocations have this characteristic, so to improve the results, the capability to target a more variable allocation would be needed. It has been

noticed, like in the plots in figure 5.1, that Transformers generate more variable allocations, so a research using these could be conducted.
Finally, as in the works of References [10] and [12] Reinforcement learning should be experimented as it models better human behaviour as desire for a reward.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[2] E. F. Fama, "Efficient capital markets: A review of theory and empirical work," *The Journal of Finance*, vol. 25, no. 2, pp. 383–417, 1970. [Online]. Available: http://www.jstor.org/stable/2325486

[3] D. Kumar, S. Meghwani, and M. Thakur, "Proximal support vector machine based hybrid prediction models for trend forecasting in financial markets," *Journal of Computational Science*, vol. 17, 07 2016.

[4] Z. Zhang, S. Zohren, and S. Roberts, "Deep reinforcement learning for trading," 2019.

[5] D. Snow, "Machine learning in asset management—part 2: Portfolio construction—weight optimization," *The Journal of Financial Data Science*, vol. 2, no. 2, pp. 17–24, 2020.

[6] Z. Zhang, S. Zohren, and S. Roberts, "Deep learning for portfolio optimization," *The Journal of Financial Data Science*, vol. 2, no. 4, p. 8–20, Aug 2020. [Online]. Available: http://dx.doi.org/10.3905/jfds.2020.1.042

[7] G. Cornuejols and R. Tütüncü, *Optimization methods in finance*. Cambridge University Press, 2006, vol. 5.

[8] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE transactions on neural Networks*, vol. 12, no. 4, pp. 875–889, 2001.

[9] J. Moody, L. Wu, Y. Liao, and M. Saffell, "Performance functions and reinforcement learning for trading systems and portfolios," *Journal of Forecasting*, vol. 17, no. 5-6, pp. 441–470, 1998.

[10] L. Weijs, "Reinforcement learning in portfolio management and its interpretation," *Erasmus Universiteit Rotterdam*, 2018.

[11] J. Y. Campbell, L. M. Viceira, L. M. Viceira *et al.*, *Strategic asset allocation: portfolio choice for long-term investors*. Clarendon Lectures in Economic, 2002.

[12] T. W. Kim and M. Khushi, "Portfolio optimization with 2d relative-attentional gated transformer," in *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. IEEE, 2020, pp. 1–6.

[13] G. Chakravorty, A. Awasthi, and B. Da Silva, "Deep learning for global tactical asset allocation," *Available at SSRN 3242432*, 2018.

[14] L. Malandri, F. Z. Xing, C. Orsenigo, C. Vercellis, and E. Cambria, "Public mood–driven asset allocation: The importance of financial sentiment in portfolio management," *Cognitive Computation*, vol. 10, no. 6, pp. 1167–1176, 2018.

[15] B. M. Rom and K. W. Ferguson, "Post-modern portfolio theory comes of age," *Journal of Investing*, vol. 3, no. 3, pp. 11–17, 1994.

[16] D. G. Luenberger *et al.*, *Investment science*. Oxford university press, 1997.

[17] G. C. Pflug, A. Pichler, and D. Wozabal, "The 1/n investment strategy is optimal under high model ambiguity," *Journal of Banking and Finance*, vol. 36, no. 2, pp. 410–417, 2012.

[18] V. Demiguel, L. Garlappi, and R. Uppal, "Optimal versus naive diversification: How inefficient is the 1/n portfolio strategy?" *Review of Financial Studies*, vol. 22, 05 2009.

[19] R. Duchin and H. Levy, "Markowitz versus the talmudic portfolio diversification strategies," *The Journal of Portfolio Management*, vol. 35, no. 2, pp. 71–74, 2009.

[20] T. Fuertes. (2017) Portfolio risk control: risk parity vs. inverse volatility. [Online]. Available: https://quantdare.com/risk-parity-versus-inverse-volatility/

[21] J. Teiletche, T. Roncalli, and S. Maillard, "On the properties of equally-weighted risk contributions portfolios," *SSRN Electronic Journal*, 09 2008.

[22] J. W. Wilder, *New concepts in technical trading systems*. Trend Research, 1978.

[23] G. Appel, *Technical analysis: power tools for active investors*. FT Press, 2005, p. 166.

[24] B. John, "Bollinger on bollinger bands," 2002.

[25] R. Kempen, "Fibonaccis are human (made)," *IFTA J*, pp. 4–9, 2016.

[26] M. Patel, *Trading with Ichimoku clouds: the essential guide to Ichimoku Kinko Hyo technical analysis*. John Wiley & Sons, 2010, vol. 473.

[27] G. W. Schwert, "Why does stock market volatility change over time?" *The journal of finance*, vol. 44, no. 5, pp. 1115–1153, 1989.

[28] A. R. Gallant, P. E. Rossi, and G. Tauchen, "Stock prices and volume," *The Review of Financial Studies*, vol. 5, no. 2, pp. 199–242, 1992.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[30] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[32] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[33] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International conference on machine learning*. PMLR, 2015, pp. 2342–2350.

[34] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.

[35] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," 2016.

[36] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] S. M. Kazemi, R. Goel, S. Eghbali, J. Ramanan, J. Sahota, S. Thakur, S. Wu, C. Smyth, P. Poupart, and M. Brubaker, "Time2vec: Learning a vector representation of time," *arXiv preprint arXiv:1907.05321*, 2019.

[39] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1–58, 01 1992.

[42] S. Salti, "Machine learning for computer vision," Course at University of Bologna, 2020.

[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[44] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87–90.

[45] R. Flanagan and L. Lacasa, "Irreversibility of financial time series: A graph-theoretical approach," *Physics Letters A*, vol. 380, no. 20, pp. 1689–1697, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0375960116002401

[46] J. B. Ramsey and P. Rothman, "Time irreversibility and business cycle asymmetry," *Journal of Money, Credit and Banking*, vol. 28, no. 1, pp. 1–21, 1996.

[47] R. A. Martin, "Pyportfolioopt: portfolio optimization in python," *Journal of Open Source Software*, vol. 6, no. 61, p. 3066, 2021. [Online]. Available: https://doi.org/10.21105/joss.03066

Appendix

Finance implications in training machine learning models

Another validation technique we considered is to invert the direction of the time dimensions, therefore doing backward forecasting. This can be seen as a form of data augmentation, to have more data to train with. This works for some kind of time series data, for example river levels, but it has been proved not suitable for financial returns data [45] [46]. Deterministic tests exist able to distinguish financial times series between originals and the one inverted temporally. This can be explained by analysing volatility, that is usually constant, aside from rare moments of volatility spikes. After those spikes, volatility descends, but more slowly than it had risen. By inverting returns, this would produce a slow increase and a sudden fall. Another test is based on the leverage effect. There is negative correlation between current returns and future volatility. Future volatility will increase much more from a current negative returns than from a current positive return. Because a negative returns is seen from investors as an increase of risk, while a positive return not. But this correlation is not existent between past volatility and future returns, this is a factor exploited by a test for distinguishing original and flipped financial returns time series.

PyPortfolioOpt

PyPortfolioOpt is a Python library that implements several portfolio optimization method and visualizations. It currently supports efficient frontier techniques and Black-Litterman allocation, as well as more recent advances in the area such as shrinkage and Hierarchical Risk Parity, as well as some innovative experimental features such as exponentially-weighted covariance matrices [47]. The library emphasize modularity: users are able to come up with their alternative models and data and feed them into the optimizer.

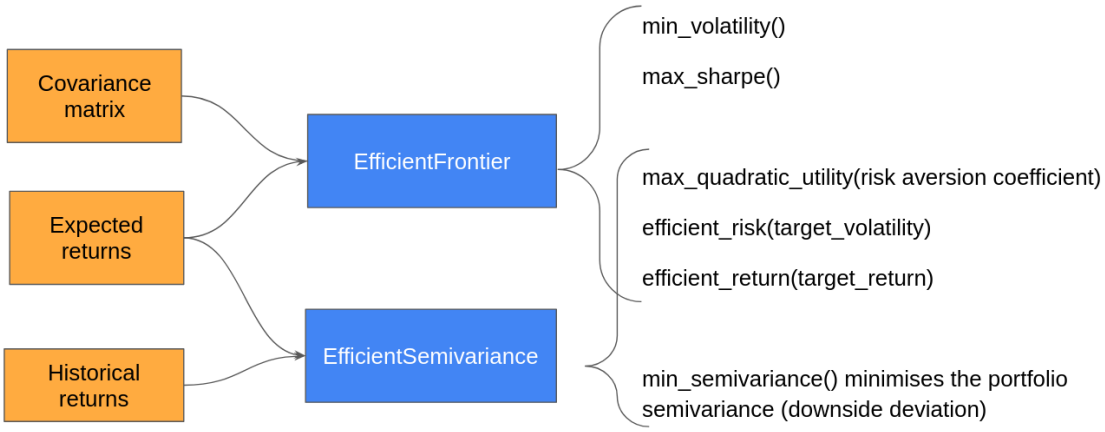


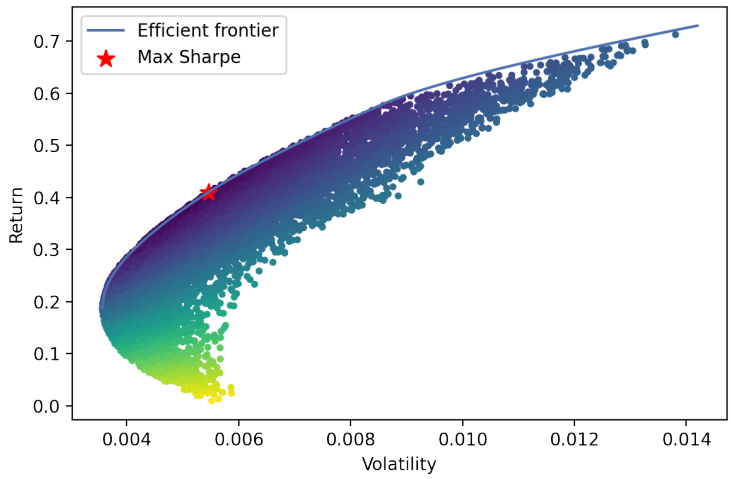
Figure 6.1: Optimization function offered by EfficientFrontier and EfficientFrontier, PyPortfolioOpt Python classes

Optimization is made starting from expected returns and assets covariance matrix. The library itself implements several methods for forecasting expected returns from simple historical mean, to exponentially weighted mean and capital asset model [16]. The covariance matrix can be obtained also from historical prices by a directed method or exponentially weighted. In this review we will show only 2 methods of optimization offered by PyPortfolioOpt, EfficientFrontier and EfficientSemivariance. Figure 6.1 is a graphic summary of the functions offered by the two classes. Some methods are common in both classes, others only of one of them.
EfficientFrontier only required the assets Covariance matrix and the Expected returns, while EfficientSemivariance computes internally the Semivariance matrix so requires the historical return and the expected returns.
The `min_volatility()`, `max_sharpe()` are straighforward and does not require any parameter. `efficient_risk` and `efficient_return` takes respectively a target volatility and target return. `min_semivariance()` minimize the portfolio semivariance as explained in chapter 2. `max_quadratic_utility` maximize the function:

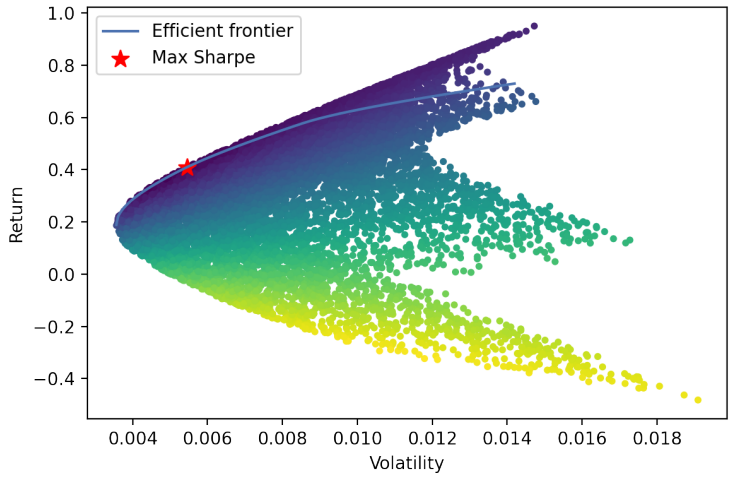
$$w^T\mu - \frac{\delta}{2}w^T\Sigma w$$

It takes as argument a risk aversion coefficient δ that must be a positive float.
PyPortfolioOpt also offers several plotting methods that help the understanding of mean-variance theory. The following figures are created from the strategy assets returns that have been used throughout this work.
Figure 2.2 is a mean-variance diagram showing the 4 assets strategy on SP500,

NASDAQ, EUROSTOXX50, NIKKEI225 and the efficient frontier position for a portfolio built with those assets.



(a) without shorting



(b) with shorting

Figure 6.2: Mean-Variance diagram: random portfolios

The coloured dots in figure 6.2 represents different, not necessary optimal, portfolios built starting from those 4 assets, respectively if short selling is allowed or not. These coloured dots are contained in the feasible region. The star dot is the position of the portfolio that maximize the Sharpe ratio.

Acknowledgements

I would like to thank first my supervisor, the Professor Fabrizio Lillo, for the great help given in writing this work. I am sure that the fact we come from different backgrounds has helped to generate qualitatively superior work. I'm a supporter of interdisciplinary collaboration, and this experience has done nothing but reinforce my opinion.
Secondly I would like to thank the dott. Nicola Donelli from Salzenberg AI for its help in introducing me in the task and practical hints during the developing.
Then, I would like to thank my friends Irene, Antonio, Rio and Maja for their motivational support in these last eight months.
Lastly but not least, i would like to thank my family for their support, that without it, I would not be here.