Time Series Forecasting with Transformer Models and Application to Asset Management*

Edmond Lezmi
Amundi Institute
edmond.lezmi@amundi.com

Jiali Xu
Amundi Institute
jiali.xu@amundi.com

February 2023

Abstract

Since its introduction in 2017 (Vaswani *et al.*, 2017), the Transformer model has excelled in a wide range of tasks involving natural language processing and computer vision. We investigate the Transformer model to address an important sequence learning problem in finance: time series forecasting. The underlying idea is to use the attention mechanism and the seq2seq architecture in the Transformer model to capture long-range dependencies and interactions across assets and perform multi-step time series forecasting in finance. The first part of this article systematically reviews the Transformer model while highlighting its strengths and limitations. In particular, we focus on the attention mechanism and the seq2seq architecture, which are at the core of the Transformer model. Inspired by the concept of weak learners in ensemble learning, we identify the diversification benefit of generating a collection of low-complexity models with simple structures and fewer features. The second part is dedicated to two financial applications. First, we consider the construction of trend-following strategies. Specifically, we use the encoder part of the Transformer model to construct a binary classification model to predict the sign of an asset's future returns. The second application is the multi-period portfolio optimization problem, particularly volatility forecasting. In addition, our paper discusses the issues and considerations when using machine learning models in finance.

Keywords: Machine learning, Transformer model, attention mechanism, seq2seq architecture, time series forecasting, quantitative asset management, backtesting, trading strategy, multi-period optimization.

JEL Classification: C45, C53, G11.

*The authors are very grateful to Amina Cherief, Thierry Roncalli and Takaya Sekine for their helpful comments. The opinions expressed in this research are those of the authors and are not meant to represent the opinions or official positions of Amundi Asset Management.

1 Introduction

The continuous increase in computer processing power and storage capacity and the explosion of data volumes have led to successful applications of deep learning models in various fields, such as computer vision, natural language processing, speech recognition, gaming and robotics. Two of the most famous are AlphaGo developed by DeepMind Technologies, the first computer program to defeat a world champion in the board game Go in 2016, and the new internet sensation ChatGPT launched by OpenAI at the end of 2022. This Chatbot can generate human-like responses to user input in a variety of situations, such as writing articles, answering questions, or generating lines of code in response to prompts.

Time series analysis has a wide range of applications in many fields, such as engineering, economics, meteorology, and finance. In particular, time series forecasting is, without a doubt, one of the most difficult tasks in investing. Due to the rapid growth of financial data, we have seen a significant increase in the number of studies on time series forecasting of financial indicators, such as macroeconomic indicators, asset prices, asset returns, and risk indicators like realized and implied volatility, etc. The traditional statistical models used to model time series can be divided into two categories: linear time series models, such as ARIMA models; and nonlinear time series models, like GARCH models. In recent years, machine learning models have had many potential applications with the ability to enhance our knowledge of financial markets.

The development of deep learning provides us with powerful tools to create the next generation of time series prediction models. Deep artificial neural networks, as a way of learning temporal dynamics in a completely data-driven way, are particularly well adapted to the challenge of finding complex nonlinear relationships between inputs and outputs. Initially, recurrent neural networks and their extended LSTM networks were designed to handle sequential information in time series. Convolutional neural networks were then used to forecast time series as a result of their success in image analysis tasks. The Transformer model was subsequently published by Google in 2017 (Vaswani *et al.*, 2017), and it was designed to use attention mechanisms to handle sequential data in order to address the sequence learning problem in natural language processing, such as machine translation. In essence, the Transformer model enables us to convert an input sequence from one domain into an output sequence from another domain. For example, we can use the Transformer model to train a robot to translate English sentences into French. By analogy, if we consider one segment of a time series as a sentence in one language, and the following segment as a sentence in another language, this multi-step time series forecasting problem is also a sequence learning problem. Therefore, the Transformer model can be used to address forecasting in time series analysis. As described in Wen *et al.* (2022), many variants of the Transformer model have been successfully applied to time series forecasting tasks, such as Li *et al.* (2019) and Zhou *et al.* (2021).

This paper is organized as follows. We begin with an overview of traditional models used for time series forecasting, and explain why the seq2seq architecture used in the Transformer model is suitable for modeling complex relationships in sequence data and for multi-step time series forecasting. In Section Three, we present the attention mechanism and the details of each part of a Transformer model. In this section, we also point out the advantages and applications of Transformer models in time series forecasting. In Section Four, we discuss the challenges of applying machine learning models in finance and argue that we should maintain a balance between model complexity, calibration quality and prediction quality. In Section Five, we show two applications of Transformer models in portfolio management: the first one is to build a trend-following strategy, and the second one concerns the multi-period portfolio optimization problem. Finally, Section Six offers some concluding remarks.

2 Multi-step time series forecasting

2.1 Sequence learning problems

We generally define a sequence as a set of symbols of the same type that are arranged in a particular order, such as a time series or a sentence. These symbols can be numbers, letters, words, or even events or objects. For example, the order in which we visit web pages is also a sequence. To the best of our knowledge, sequence learning problems include at least 3 categories as follows:

- **Sequence prediction**
  The objective is to predict a categorical label or a continuous value. For example, given a financial time series, we often predict the next value of the sequence itself, which is also known as time series forecasting;
- **Sequence generation**
  The objective is to convert sequences from one domain into sequences from another domain, such as machine translation, text summarization, chatbots, etc.;
- **Sequential decision making**
  The objective is to make various decisions that are sequential in order to optimize the whole process, such as optimally playing a card game.

It is worth mentioning that the boundary between the above categories is not strict, which means that one problem can be classified into different categories. For instance, chatbots can also be categorized as both a sequence generation problem and a sequential decision making problem, i.e., the bot must not only generate appropriate answers to the questions, but also take the context into account in order to give the best answer.

2.2 Traditional methods for multi-step time series forecasting

In the field of economics and finance, we are most exposed to time series consisting of numbers ordered in time, and we are mainly concerned with sequence prediction problems, such as predicting national GDP, forecasting financial market returns and volatility, etc. For a long time, we have been using traditional statistical models to model time series, such as autoregressive integrated moving average model (ARIMA), generalized autoregressive conditional heteroskedasticity model (GARCH), etc. With the rapid development of computing power and the popularity of machine learning over the past few years, more and more machine learning models, especially deep learning models, are applied to time series forecasting, such as recurrent neural networks (RNNs), long short-term memory model (LSTM), gated recurrent unit model (GRU), etc.

Let $X_1^t = \{x_1, x_2, \ldots, x_t\}$ be a time series of t past values of a dynamic system, where each x_i is a d-dimensional vector, i.e. $x_i = (x_i^1, x_i^2, \ldots, x_i^d)$. The time series forecasting task at time t is to predict the values $X_{t+1}^{t+\tau} = \{x_{t+1}, x_{t+2}, \ldots, x_{t+\tau}\}$ at the τ future time steps. When $d = 1$, we have an univariate time series forecasting problem and when $d > 1$, the problem is multivariate. Furthermore, when $\tau > 1$, we call the task multi-step time series forecasting and we have two modeling approaches:

- **Iterated multi-step forecasting**
  In this case, our goal is to build a single-step forecaster f that models between t past values $X_1^t = \{x_1, x_2, \ldots, x_t\}$ and the next future value x_{t+1} of the dynamic system:

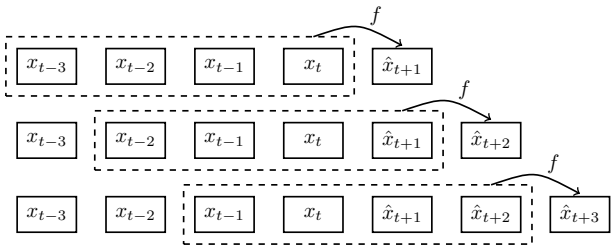
$$x_{t+1} = f\left(X_1^t\right) + \epsilon$$

where ϵ is the error term, often called noise. In the process of prediction, we iteratively apply this model to obtain multi-step predictions as shown in Figure 1.

$$\begin{aligned}\hat{x}_{t+1} &= f\left(x_1, x_2, \ldots, x_t\right)\\ \hat{x}_{t+2} &= f\left(x_2, x_3, \ldots, \hat{x}_{t+1}\right)\\ \hat{x}_{t+3} &= f\left(x_3, x_4, \ldots, \hat{x}_{t+2}\right)\\ &\ldots\end{aligned}$$

Thus, we only need to build one model f to make multi-step predictions, but the error will keep accumulating because the prediction at time $t+1$ is used as input for the next time step.

Figure 1: Iterated multi-step forecasting



Source: Amundi Institute.

- **Direct multi-step forecasting**
  Alternatively, we can look for a series of models g_i with $i = 1, 2, \ldots, \tau$, each modeling the relationship between t past values $X_1^t = \{x_1, x_2, \ldots, x_t\}$ and one future value of the dynamic system. It follows that:

$$\begin{aligned}x_{t+1} &= g_1\left(X_1^t\right) + \epsilon_1\\ x_{t+2} &= g_2\left(X_1^t\right) + \epsilon_2\\ &\ldots\\ x_{t+\tau} &= g_\tau\left(X_1^t\right) + \epsilon_\tau\end{aligned}$$

  where ϵ_i is a series of noises. Furthermore, using the property that neural networks support multiple outputs, we can replace all single models g_i with a model g with more parameters, which can predict all τ future values at once:

$$X_t^{t+\tau} = g\left(X_1^t\right) + \epsilon$$

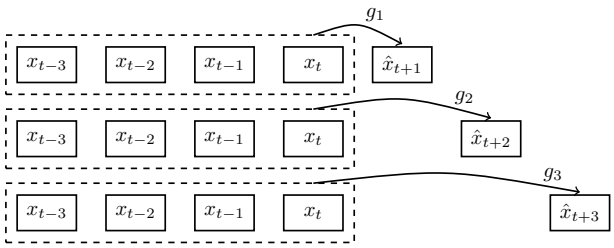
  The process of prediction is shown in Figure 2. In this case, we directly predict the values of several time steps without accumulating errors, but we do not model the relationship between $x_{t+1}, x_{t+2}, \ldots, x_{t+\tau}$. In addition, this approach requires training more models or a model with more parameters.

2.3 Seq2seq models

As mentioned in the previous section, an increasing number of deep learning models have recently been used to process sequential data, the best known of which are RNN and LSTM[1].

[1]The technical details about RNN and LSTM models can be found in Appendix B.1 and B.2

Figure 2: Direct multistep forecasting

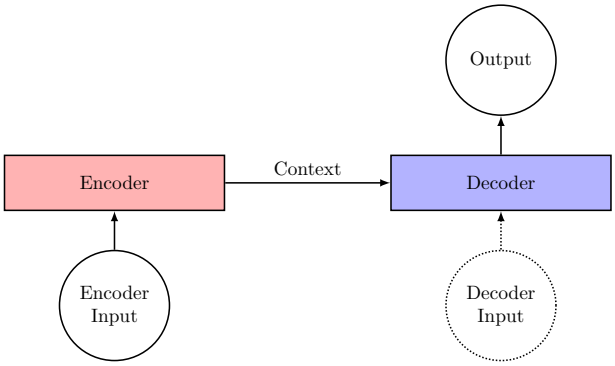


Source: Amundi Institute.

However, the structure of classical RNN or LSTM models is not flexible enough and cannot model very complex relationships in sequence data. In some NLP tasks, such as machine translation, speech recognition, text summarization and question and answer systems, the lengths of the input and output sequences are often not the same. In this case, we need another structure of neural networks, namely the sequence-to-sequence model (seq2seq model), which can solve these problems well. In short, seq2seq models consist of training models that convert sequences from one domain into sequences from another domain. As shown in Figure 3, a seq2seq model, which can be considered as a special type of many-to-many structure, usually has an encoder-decoder structure:

- **Encoder**: it transforms the input sequence into a single fixed length vector, called the context vector. This context vector contains all the information that the encoder can extract from the input, including relationships between different time steps and within network layers.
- **Decoder**: it transforms the context vector into the output sequence. Depending on the objectives and different ways of training, we may have input for the decoder or not.

Figure 3: The structure of seq2seq model



Source: Amundi Institute.

Due to the increasing computer performance since 2000, more and more deep learning models are used to deal with time series forecasting problems, since these problems can be

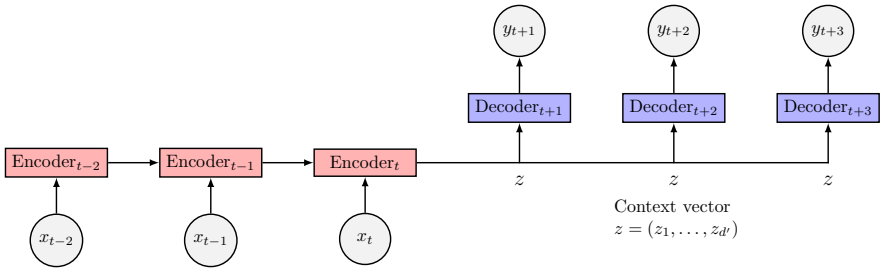
considered as sequence-to-sequence problems, especially in the case of multi-step time series forecasting. In practice, the encoder and the decoder can consist of a single or a stack of RNN layers and LSTM layers.

The encoder will transform an input sequence $\{x_1, \ldots, x_t\}$ into a d'-dimensional context vector $z = (z_1, \ldots, z_{d'})$ and the decoder will map this sequence of context to an output sequence $\{y_{t+1}, \ldots, y_{t+\tau}\}$. As we have explained in Section 2.2, two main methods can be used for multi-step time series forecasting when using traditional models such as ARIMA or GARCH. Similarly, we can use the encoder-decoder structure in two ways:

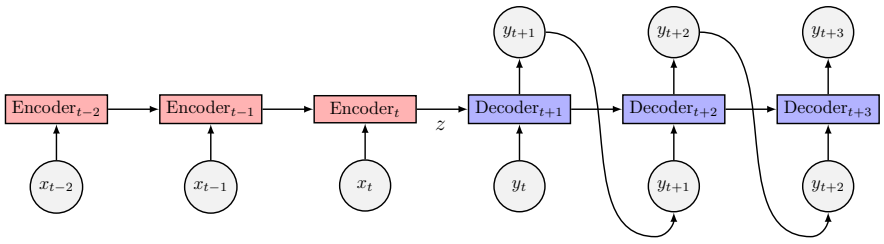
- Figure 4a illustrates a simple encoder-decoder structure that corresponds to the direct multi-step forecasting in Figure 2. In this structure, the context vector z is common to all outputs $\{y_{t+1}, \ldots, y_{t+\tau}\}$, and there are no other inputs for the decoder.
- Figure 4b illustrates a recursive encoder-decoder structure, which is similar to the iterated multi-step forecasting. The context vector z is used only as input to Decoder_{t+1} to predict the first output y_{t+1}, then z may be updated before being passed to Decoder_{t+2}. In particular, we may have inputs for the decoder in this structure. If we do not use the ground truth labels as the input of the decoder during training, we call it free running technique and if we use them as the correct answer, the approach is called teacher forcing technique. More technical details about free running and teacher forcing techniques can be found in Appendix B.3.

Figure 4: The illustration of Encoder Decoder

(a) Simple encoder-decoder



(b) Recursive encoder-decoder



Source: Amundi Institute.

The use of an encoder-decoder structure with the teacher forcing technique is suitable for multi-step time series forecasting problems. In this case, we can model the dependence relationship between predictions while also mitigating forecast error accumulation.

3 Transformer model

In many sequence learning problems, such as machine translation, understanding the context is crucial. As shown in Example 1, the word "it" refers to different things in these two similar sentences, depending on the context. In the first sentence, the word "it" refers to the dog, and in the second sentence, the word "it" means the bone.

Example 1.
- *The dog didn't eat the bone because it wasn't hungry.*
- *The dog didn't eat the bone because it smelled bad.*

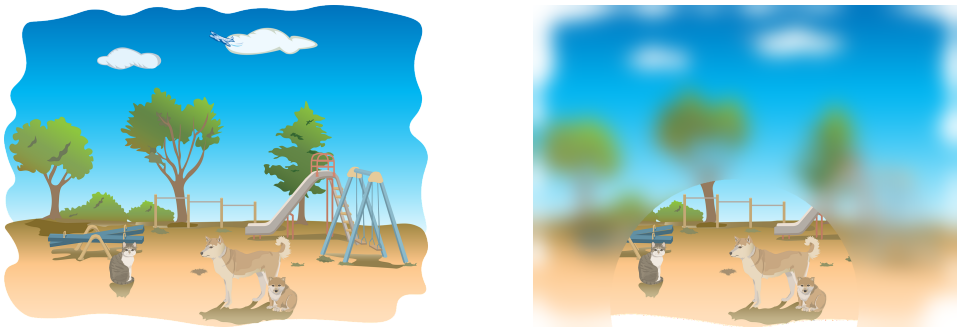
Sometimes the valuable information appears early in the sequence, so it is difficult for the model to capture it, especially in sequence prediction problems. To address this challenge, Google introduced the Transformer model in Vaswani *et al.* (2017), which is an evolution of the seq2seq model. Transformer models are designed to use attention/self-attention mechanisms to capture patterns and long-term memory in the data, such as dependencies between words in a paragraph or contextual information in an article. The most significant difference between the attention mechanism and traditional RNN or LSTM models is that the attention mechanism focuses directly on specific parts of the sequence rather than treating them equally according to order, which makes it possible for the model to capture information at very early positions in the sequence. As a result, the attention mechanism will help the model to have a better understanding of the context of the sequence. By analogy, the "context" is also very important in the financial market, such as serial correlation, volatility clustering, regime switching, and some significant financial events. This is the driving force behind our desire to apply a Transformer model to time series forecasting problems in portfolio management. In this section, we will introduce the different parts of a Transformer model and their uses and benefits, and in the next section, we will show that the use of Transformer models can provide computational advantages in some quantitative investment applications.

3.1 Attention mechanism

When we look at a picture, we are attracted to certain conspicuous parts of the picture and ignore others, which means that our attention is not equally distributed to all areas of the picture. As shown in Figure 5, when we first look at the painting, we focus more on the dogs and cats than on the trees and slides, and, in particular, most people would completely ignore the plane in the sky. Mathematically speaking, we give different weights to different areas of the picture: areas with high weights will produce a higher intensity of signals, and these signals will be received by the brain through the eyes. Thus, the attention mechanism is in fact a weight distribution system. In practice, we want to mimic this attention mechanism of our brain in deep learning so that the model can pay more attention to the important parts of the input data, depending on the context.

3.1.1 Queries, keys and values

Vaswani *et al.* (2017) introduced an attention mechanism called scaled dot-product attention in their Transformer models. In their paper, the authors used the query (Q) / key (K) / value (V) concept that is often found in information retrieval systems: given a query, we calculate the relevance of the query to the key, and then find the most suitable value based on the relevance. For instance, when we do online shopping at an e-commerce platform, we will enter our preferences, such as brand, price, features into the search engine. The site will then match our preferences to all available products' basic details and return the one that



(a) Animals in a playground (b) Our attention is drawn to animals

Figure 5: An illustration of the attention mechanism

Source: pixabay.com & Amundi Institute.

best matches our preferences. In this example, the query is our preference, the key is the basic information of each product and the value is the product. By analogy, the attention mechanism in Transformer models uses the dot-product[2] to calculate the similarity between the query and the key, and uses a softmax function[3] to decide the weights of each element of values. Higher weights will be assigned to those elements whose corresponding keys are more relevant to the query.

Dot-product is a simple way to measure the relevance of the query to the key, but this operation requires that both the query and the key have the same dimension. Let d_k denote the dimension of the query and the key, and d_v denote the dimension of the value. Formally, the attention vector for n queries Q and m key-value pairs (K, V) is calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}_k\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{1}$$

where $Q \in \mathbb{R}^{n\times d_k}$, $K \in \mathbb{R}^{m\times d_k}$, $V \in \mathbb{R}^{m\times d_v}$ and we can notice that the dot products $QK^\top$ are scaled down by $\sqrt{d_k}$. This is also the origin of the name "*scaled dot-product attention*" for Equation (1). We may suppose that Q and K have a mean of 0 and a variance of 1 in each dimension. In this case, the matrix multiplication $QK^\top$ will have a mean of 0 and variance of d_k in each dimension. So the square root of d_k is used to ensure the variance of $QK^\top$ is scaled to one, regardless of the value of d_k. The consistent variance of the matrix multiplication facilitates the training process of machine learning models.

In addition, we notice that the dot product between queries Q and keys K, which determines the attention vector, is a type of kernel function. As explained in Tsai *et al.* (2019), we can reformulate Equation (1) via the lens of kernel. For the i-th row of $\text{Attention}(Q, K, V)$,

[2]The dot product of two vectors is equal to the product of their magnitudes and the cosine of the angle between them. A larger dot product means a smaller angle. Therefore, we may use the dot product to determine the similarity between two vectors.

[3]The softmax function is a generalization of the logistic function to multiple dimensions. It converts a numerical vector into a probability vector, where the probability is proportional to the exponentials of the numerical value. Given $z = (z_1, \ldots, z_d) \in \mathbb{R}^d$, the standard softmax function $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ is defined by the formula:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} \text{ for } i = 1, \ldots, d$$

we have:

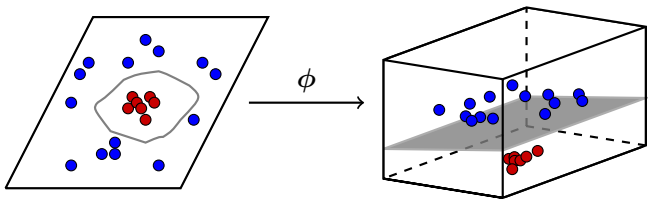
$$\begin{aligned}\text{Attention}(Q,K,V)_i &= \sum_{j=1}^{m} \frac{\exp\left(\frac{Q_i K_j^\top}{\sqrt{d_k}}\right)}{\sum_{j'=1}^{m} \exp\left(\frac{Q_i K_{j'}^\top}{\sqrt{d_k}}\right)} V_j \\ &= \sum_{j=1}^{m} \frac{\mathcal{K}\left(Q_i, K_j\right)}{\sum_{j'=1}^{m} \mathcal{K}\left(Q_i, K_j\right)} V_j\end{aligned} \tag{2}$$

where Q_i, K_j are respectively the i-th row of Q and the j-th row of K, and $\mathcal{K}(\cdot,\cdot) : \mathbb{R}^{d_k} \times \mathbb{R}^{d_k} \to \mathbb{R}$ is the kernel function defined as:

$$\mathcal{K}(x,y) = \langle \phi(x), \phi(y) \rangle$$

where x, y are d_k-dimensional inputs, $\langle \cdot,\cdot \rangle$ denotes the inner product and $\phi(\cdot)$ is a map from d_k-dimension to another dimension space, usually a higher dimension space. It makes sense in machine learning to transform the input feature space into a higher dimensional space. For example, we can project non-linear separable data onto a higher dimension space via a map $\phi(\cdot)$, so as to make it easier to classify the data with a hyperplane, as illustrated in Figure 6.

Figure 6: An illustration of the benefit of mapping to a higher dimension space



Source: Amundi Institute.

However, because it involves operations in higher dimensional spaces, it can be rather expensive to first compute $\phi(x), \phi(y)$, then $\langle \phi(x), \phi(y) \rangle$ to produce just a scalar value for $\mathcal{K}(x,y)$. The kernel trick is a technique that allows us to determine directly an explicit kernel function $\mathcal{K}(x,y)$ without computing the coordinates of the data in a higher dimensional space, which means that we compute $(x,y) \to \mathcal{K}(x,y)$ directly without passing $\phi(x), \phi(y)$ and $\langle \phi(x), \phi(y) \rangle$.

Example 2. *Kernel trick for a 2nd-degree polynomial mapping*
We assume that $x = (x_1, x_2)$, $y = (y_1, y_2)$ *and* $\phi(x) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$. *Then, we have* $\langle \phi(x), \phi(y) \rangle = x_1^2y_1^2 + 2x_1y_1x_2y_2 + x_2^2y_2^2 = \left(x^\top y\right)^2$. *Therefore, in order to simplify the calculation, we can define* $\mathcal{K}(x,y) = \left(x^\top y\right)^2$.

In addition, Equation (3) is a class of linear smoother with kernel smoothing. In the case of a non-negative kernel function $\mathcal{K}(\cdot,\cdot)$, the attention vector is a weighted average of all values of V and we can rewrite this equation in the form of the expected value:

$$\begin{aligned}\text{Attention}(Q,K,V)_i &= \sum_{j=1}^{m} \frac{\mathcal{K}\left(Q_i, K_j\right)}{\sum_{j'=1}^{m} \mathcal{K}\left(Q_i, K_j\right)} V_j \\ &= \mathbb{E}_{\mathbb{P}(K_j|Q_i)}(V)\end{aligned} \tag{3}$$

where $\mathbb{P}\left(K_j|Q_i\right) = \frac{\mathcal{K}\left(Q_i,K_j\right)}{\sum_{j'=1}^{m}\mathcal{K}\left(Q_i,K_j\right)}$ is the conditional probability function determined by the kernel function $\mathcal{K}\left(\cdot,\cdot\right)$. Tsai *et al.* (2019) demonstrate how these kernel functions help us understand the Transformer model's attention mechanism in an unified way. Different choices for the kernel function imply various Transformers architectures.

3.1.2 Self-attention

There are various attention mechanisms depending on the different definitions of queries, keys and values. In the Transformer model introduced by Vaswani *et al.* (2017), the authors use a self-attention mechanism, which means that the query, key and value come from the same sequence. Given a multivariate sequence $X \in \mathbb{R}^{n\times d_{\text{input}}}$, such as a sentence of length n converted to a word embedding representation of dimension d_{input}, we use d_{model} to denote the dimension of the query, key and value and we have:

$$
\begin{aligned}
Q &= XW^Q \\
K &= XW^K \\
V &= XW^V
\end{aligned}
$$

$$
\text{Self-Attention}(X) = \text{softmax}\left(\frac{XW^Q\left(XW^K\right)^\top}{\sqrt{d_{\text{model}}}}\right)XW^V \tag{4}
$$

where W^Q, W^K and $W^V \in \mathbb{R}^{d_{\text{input}}\times d_{\text{model}}}$ are parameter matrices to learn and have the same dimension, such that the queries Q, keys K, and values V are all $n \times d_{\text{model}}$ matrices. We train the model with the aim of learning these parameters W^Q, W^K and W^V that will transform the input X to queries Q, keys K and values V.

Self-attention is designed to capture the dependencies in the sequence, such as the relationship between each word with each other word in a sentence, which is the focus of sequence learning. With the help of neural networks, the model will learn the attention distribution of each element relative to the other elements in the same sequence. As shown in Example 1, for the word "it", in the first sentence, more attention weight should be assigned to the words like"dog" and "hungry", and in the second sentence, the words such as "bone", "smell" and "bad" should have higher attention weight.

According to Vaswani *et al.* (2017), self-attention has several advantages, such as fewer parameters and a better ability to capture long-term memory in sequences, which is a key challenge for many sequence learning problems. In the self-attention mechanism, elements in a sequence have practically no notion of order, which makes it easier for the model to learn long-range dependencies. The model will directly learn the attention weight of an element relative to all other elements, even those that are far away from it.

3.1.3 Multi-head attention

As we have explained in the previous paragraph about the self-attention mechanism, for a given query, we compare it with all keys K and get different weights for different values V. The weights can be interpreted as the relevance between the query and each key. Then, we use these weights to weight the values and average them to obtain an attention score. Therefore, our task is to learn W^Q, W^K and W^V using neural networks. Once these parameters are learned, the queries, keys, values are fixed, then the assignment of weights and the attention score matrix are also fixed. However, we want to increase the flexibility of the model and be able to understand the information in the sequence from different

aspects. To address this, we will do the scaled dot-product attention process several times independently to obtain different attention score matrices and then merge them together. In this case, we refer to each scaled dot product attention as a head and the whole process is known as the multi-headed attention mechanism.

Let h denote the number of head in the multi-head self-attention mechanism. Given an input X, we have:

$$\begin{aligned} Q_i &= XW_i^Q \\ K_i &= XW_i^K \qquad \text{for } i = 1, \cdots, h \\ V_i &= XW_i^V \end{aligned}$$

where $W_i^Q \in \mathbb{R}^{d_{\text{input}} \times d_{\text{head}}}, W_i^K \in \mathbb{R}^{d_{\text{input}} \times d_{\text{head}}}, W_i^V \in \mathbb{R}^{d_{\text{input}} \times d_{\text{head}}}$. Then, each head is an attention score matrix:

$$\text{head}_i = \text{Attention}\left(Q_i, K_i, V_i\right)$$

Then, we concatenate all these attention score matrices:

$$\text{Self-MultiHead}(X) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_\text{h}\right) W^O \tag{5}$$

where $W^O \in \mathbb{R}^{h \cdot d_{\text{head}} \times d_{\text{model}}}$ is also a learnable parameter for linear transformation.

As the authors state in their paper Vaswani *et al.* (2017), "*multi-head attention allows the model to jointly attend to information from different representation subspaces at different position*", which means that the model can use different ways to learn the dependencies in the sequence, such as capturing both short-term and long-term dependencies. Based on this design, more complex functions than a simple weighted average can be expressed by a multi-head attention mechanism. Moreover, it can be used as ensembles and play the role of aggregation to avoid over-fitting and allows us to perform parallel computing to speed up the learning process.

3.1.4 Positional encoding

As mentioned in Sections 3.1.2 and 3.1.3, the self-attention and multi-head attention are permutation-equivariant with respect to its inputs, which means that the result will not change as we change the order of elements in a sequence, due to the fact that we only do dot product operations in the attention mechanism. Thus, any element in the sequence has the same distance from all other elements, so that the model will capture long-term dependencies more easily. However, for some sequence learning problems, such as machine translation, we do not want to completely ignore the positional information in sequences. To address this, we add positional encoding to the input features.

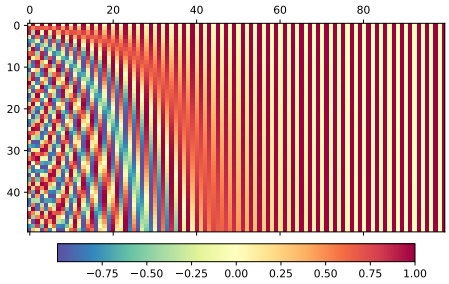
$$\tilde{X} = X + PE$$

To the best of our knowledge, there is no state-of-the-art model among many choices of positional encoding, but some criteria need to be met, such as positional embedding can reflect the positional relationship of elements of the sequence, the values of elements of positional encoding should not too large compared to the input features and should be deterministic, etc. In Vaswani *et al.* (2017), the authors designed a positional encoding using sine and cosine functions of different frequencies:

$$PE = \begin{cases} p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d_{\text{input}}}}\right) \\ p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d_{\text{input}}}}\right) \end{cases}$$

where $j = 0, 1, \cdots, d_{\text{input}}/2 - 1$. Gehring *et al.* (2017) also claims that we can also use the learned positional encoding, and according to their tests, there is essentially no difference in the results obtained by these two versions of positional encoding.

Figure 7: The illustration of the positional encoding



Source: Amundi Institute.

Machine learning methods are often unable to deal with text data directly, so it is necessary to find suitable methods to convert text data into numeric data, which leads to the concept of word embedding. Mathematically, an embedding is a relatively low-dimensional space into which a high-dimensional vector may be transformed. The embedding technique serves as a way of doing feature extraction and dimensionality reduction as well as making it easier to perform machine learning on large inputs. Therefore, the usage of embedding techniques is a key success factor for many applications in Deep Learning, such as:

- In the field of computer vision, a convolutional neural network (CNN) uses kernels to create the image embedding and pull out features like texture and edges from the raw image.

- Word2Vec (Mikolov *et al.*, 2013), which is a type of word representation that allows words with similar meaning to have a similar representation, is one of the most used embedding techniques in natural language processing.

- In graph neural networks, node embedding techniques such as DeepWalk (Perozzi *et al.*, 2014), Node2vec (Grover and Leskovec, 2016) and graph embedding techniques like Graph2vec (Narayanan *et al.*, 2017) are frequently employed.

Though widely used in Deep Learning, embedding techniques are less prevalent when applied to time series because it is difficult to measure how similar the samples are since alignment is required. One of the embedding techniques often used is the Time2Vec model, which was first developed by Kazemi *et al.* (2019). For a raw time series τ, Time2Vec of τ, denoted as $\mathbf{t2v}(\tau)$, is a vector of size $k+1$ defined as follows:

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \tau + \varphi_i, & \text{if } i = 0 \\ \mathcal{F}\left(\omega_i \tau + \varphi_i\right), & \text{if } 1 \leq i \leq k \end{cases}$$

where $\mathbf{t2v}(\tau)[i]$ is the i-th element of $\mathbf{t2v}(\tau)$, $\mathcal{F}$ is a periodic activation function, and ω_i and φ_i are learnable parameters. Therefore, this technique can also be utilized as a learnable positional encoding in the Transformer model when applied to time series data.

3.2 Model architecture

As explained by Vaswani *et al.* (2017), the network architecture of a Transformer model follows a recursive encoder-decoder structure with multi-head attention mechanisms, using teacher forcing technique for training. It is actually equivalent to the parallelized version of the recursive encoder decoder as shown in Figure 4b. Therefore, we need to provide inputs to the encoder and decoder, as shown in Figure 9. If we take a machine translation task as an example, during the training process, each time the input to the encoder is a complete English sentence, the input to the decoder will be the corresponding French sentence but shifted right and the output of the decoder will be the next word in this French sentence, as shown in Figure 8.

Figure 8: The illustration of the inputs and outputs of the Transformer model

Encoder Inputs	**Decoder Inputs**	**Decoder Outputs**
The coffee is free here	Le	café
The coffee is free here	Le café	est
The coffee is free here	Le café est	gratuit
The coffee is free here	Le café est gratuit	ici
The coffee is free here	Le café est gratuit ici	<end>

Source: Amundi Institute.

3.2.1 Encoder

The task of the encoder is to map an input sequence to a continuous representation called a "context" vector. In a Transformer model, the encoder part consists of a stack of N identical encoder blocks, all of which are connected back-to-back (series connection). Thus, N controls the depth of the network. Obviously, a larger N means a more complex model and more parameters to learn. Each encoder block, as shown on the left side of Figure 9, is made up of two parts:

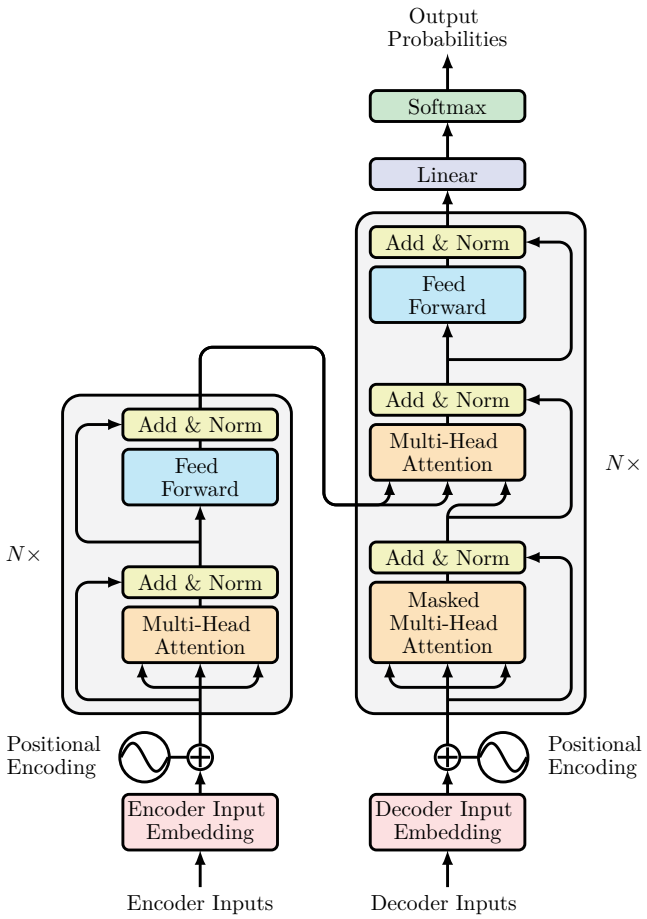
- **A multi-head self-attention layer**
  This layer, as described in Section 3.1.3, measures the relevance of each element in relation to all other elements in the sequence.
- **A fully connected feed-forward network of two layers**
  The first layer takes ReLU[4] as the activation function and the second takes Linear[5] activation function, so that :

$$\mathrm{FFN}(x) = \mathrm{ReLU}(W_1 x + b_1) W_2 + b_2$$

where W_1, b_1 and W_2, b_2 are respectively the weights and bias parameters of two layers. This fully connected feed-forward network acts as the hidden layers in a classical neural network.

[4]Rectified linear unit (ReLU) : $f(x) = x^+ = \max(0, x)$
[5]Linear : $f(x) = x$

Figure 9: The Transformer model architecture taken from "Attention is All Your Need" Vaswani *et al.* (2017)



Source: Vaswani *et al.* (2017).

In addition, we apply residual connection and layer normalization to these two components, respectively. The residual connection is to solve the degradation problem in deep learning, as explained in Appendix B.5, while the layer normalization technique is to normalize the statistics of the output of the hidden layers to avoid the vanishing or exploding gradient problem due to high learning rates. We will find more technical details about the layer normalization technique in Appendix B.4. Thus, let X denote the input of the decoder, then we have in each encoder block:

$$\begin{aligned}
\tilde{X} &= X + PE && \text{(positional encoding)} \\
\tilde{X} &= \text{LayerNorm}\left(\tilde{X} + \text{Self-MultiHead}(\tilde{X})\right) && \text{(multi-head self-attention)} \\
\tilde{X} &= \text{LayerNorm}\left(\tilde{X} + \text{FFN}\left(\tilde{X}\right)\right) && \text{(feed-forward network)}
\end{aligned}$$

3.2.2 Decoder

As shown on the right side of Figure 9, the decoder part also consists of a stack of identical decoder blocks and each decoder block has 3 components:

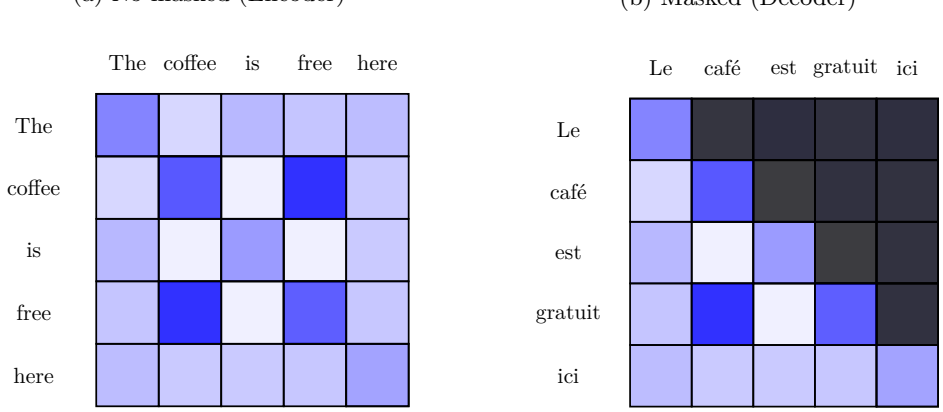
- **A masked multi-head self-attention layer**
  This layer is similar to the one in the encoder block, and its purpose is also to capture the relationship between the elements of the decoder sequence input. In the multi-head attention mechanism, all inputs will be fed together and linear transformations will be done in each head to capture the connections between data according to different criteria. It is no problem for the encoder to receive all the inputs at once. In Figure 10a, we illustrate an example of the attention score matrix for the multi-head self-attention mechanism in the encoder. For the word "*free*", it is quite normal to pay more attention to the word "*coffee*", which allows us to understand that "*free*" in this sentence means "*without cost*" and not "*no longer restricted*". However, the Transformer model's decoder still predicts the output sequence recursively, meaning that the decoder predicts one element at a time, based on the elements generated so far. To ensure that future information is not utilized, we must hide the sequence of upcoming elements during the training process. So, we need to add a padding mask to the input for the decoder as shown in Figure 10b. We call this process masked forward looking.

- **A multi-head attention layer**
  This part is still different from the multi-head self-attention layer in the encoder block and the masked multi-head self-attention layer in the decoder block. In this multi-head attention layer, the query, the key and the value come from different sources: the query comes from the previous component in the decoder block and the key and value are the output of the encoder. In fact, this mechanism is responsible for searching the relationship between the input of the encoder and the input of the decoder.

- **A fully connected feed-forward network of two layers**
  This part is identical to the one implemented in the encoder block.

We add residual connections and a normalization layer for each component in the decoder block, just as we did in the encoder block. Thus, let Y be the input of the decoder, then we

Figure 10: Multi-head self-attention in the Transformer model



Source: Amundi Institute.

have in each decoder block:

$$\tilde{Y} = Y + PE \qquad \text{(positional encoding)}$$

$$\tilde{Y} = \text{LayerNorm}\left(\tilde{Y} + \text{Masked Self-MultiHead}(\tilde{Y})\right) \qquad \text{(masked multi-head self-attention)}$$

$$\tilde{Y} = \text{LayerNorm}\left(\tilde{Y} + \text{MultiHead}(\tilde{Y}, \tilde{X}, \tilde{X})\right) \qquad \text{(multi-head attention)}$$

$$\tilde{Y} = \text{LayerNorm}\left(\tilde{Y} + \text{FFN}\left(\tilde{Y}\right)\right) \qquad \text{(feed-forward network)}$$

where $\tilde{X}$ is the output of the encoder as described in Section 3.2.1.

3.3 Advantages of Transformer models

In summary, Transformer models use the seq2seq architecture, and its flexibility allows us to handle more complex sequence learning problems. Using the attention mechanism, we can capture the long-term dependencies between the elements in the sequence, and in particular, using multi-head attention will capture the information in the sequence from different aspects. Furthermore, in addition to the self-attentive mechanisms we apply for the encoder and decoder, we use another attention mechanism to capture the relevance between the encoder and decoder. Since the self-attentive mechanisms do not analyze their inputs in chronological order, the Transformer model is less likely to suffer from the vanishing or exploding gradient problem[6]. Using teacher forcing technique in the training process will also avoid the accumulation of errors on multi-step prediction problems or sequence generation problems. In addition, another advantage of Transformer models is their parallelization. Each head in a Transformer model can capture the relationship of the elements in the input with all other elements on different standards thanks to the multi-head attention mechanism. As a comparison, we need to feed the elements into the RNN model one by one in temporal order, which makes its parallelization impossible.

[6]More technical details can be found in Appendix B.1

3.4 Applications to time series forecasting

By understanding the principle of the attention mechanism and the seq2seq architecture of Transformer models, we find that they are very suitable for sequence generation tasks, where the goal is to convert sequences from one domain into sequences from another domain. Similarly, we can also apply these advanced machine learning techniques to time series forecasting, especially multi-period multivariate time series forecasting, as shown in Figure 11. In this type of task, we need to learn not only the temporal relationships in time series to model how dynamic systems evolve, but also the spatial relationships in multivariate data to understand how they affect each other. In finance, time series forecasting is a common task and plays a very important role in analyzing the economy and making business decisions. Therefore, we will illustrate two financial applications of Transformer models:

- **Using only the encoder part to make one-step prediction**

  As illustrated in Figure 12, if we use only the encoder part of the Transformer model and connect the encoder output directly to the final layer, the model is similar to a traditional many-to-one type[7] of RNN but employs an self-attention mechanism. Therefore, we can use this model in one-step prediction problems, as we often do with recursive models like RNN or LSTM and these recursive models can be completely replaced by encoders because it allows for more flexible parallelization, more efficient long-term memory retention, and fewer vanishing or exploding gradient problems. We can also handle different problems by modifying the activation function in the last layer of the model, such as classification problems, e.g., predicting the sign of next week's return, or dealing with regression problems, e.g., single-step time series forecasting.

- **Using the complete Transformer model to make multi-step prediction**

  As explained in Section 2.2, the one-step prediction model is difficult to apply to multi-step prediction tasks, whether using the iterative approach shown in Figure 1 or the direct approach shown in Figure 2. Because of the seq2seq architecture in Transformer models, we can use these models to handle a multi-step prediction problem.

Figure 11: Inputs and outputs of a Transformer model for time series forecasting

Encoder Inputs	**Decoder Inputs**	**Decoder Outputs**
x_{t-5} x_{t-4} x_{t-3} x_{t-2} x_{t-1}	x_t	x_{t+1}
x_{t-5} x_{t-4} x_{t-3} x_{t-2} x_{t-1}	x_t x_{t+1}	x_{t+2}
x_{t-5} x_{t-4} x_{t-3} x_{t-2} x_{t-1}	x_t x_{t+1} x_{t+2}	x_{t+3}

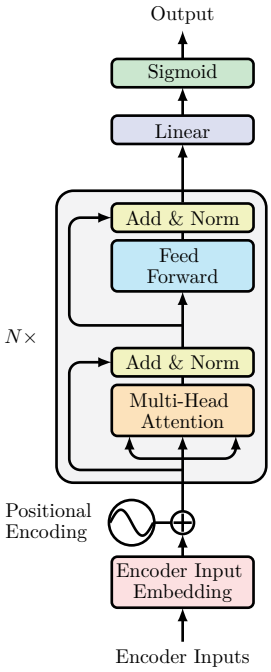
Source: Amundi Institute.

Therefore, in Section 5, we will test Transformer models in two separate situations: time-series trend-following strategy and multi-period portfolio optimization problem.

- In the case of a trend following strategy, we will use the encoder part of the Transformer model to predict the sign of an asset's next week's return using its short, medium, and

[7] Figure 21 in Appendix B.1.

Figure 12: One-step prediction model using only the encoder part



Source: Vaswani *et al.* (2017) & Amundi Institute.

long term trends. In our case, we do not put all these features together to train a complex model. We borrow the idea of weak learners in ensemble learning, and we will train many models, each simple and using only one or two features. Finally, we will aggregate these weak learners to get a strong learner. In Section 4.1, we will explain our motivation for doing so.

- For the multi-period portfolio optimization problem, we will use the full Transformer model to make multi-period forecasting of volatility time series and these forecasts will be used as inputs to portfolio allocation methods, such as the mean variance optimization approach and the risk budgeting approach. We will compare the differences between single-period optimal portfolios, multi-period optimal portfolios based on historical estimates and multi-period optimal portfolios based on Transformer's forecasts.

4 Quantitative investment and machine learning

Due to the vast increase in computational power, more and more machine learning models have been applied to the financial field, enabling us to better understand financial markets. In the investment field, one of the most difficult and important tasks in portfolio construction is the forecasting of future returns. We also believe that machine learning models may bring some computational advantages to quantitative investing, such as the ability to capture non-linear relationships and time dependence. For instance, Cherief *et al.* (2022) have employed tree-based machine learning algorithms to capture nonlinearities and detect interaction effects among risk factors in the EUR and USD credit space. However, in our opinion, successful application of a machine learning model requires many conditions to

be met: the data must be clean, the amount of data must also be sufficient, the patterns present in the data must be persistent, the complexity of the model must be appropriate, and the quality of the training must also be very good. In practice, we often cannot meet all the previous conditions due to the differences between financial time series data and data in other domains, such as text or images. In this case, machine learning models and training processes must be carefully designed to prioritize conditions that are also important in quantitative investing, such as diversification. Therefore, we believe that using machine learning in quantitative investing is a big engineering problem that requires careful attention at every step of the application process.

4.1 The challenges of applying machine learning in finance

Machine learning models are sometimes perceived as black boxes, which means that the algorithms give you a result without explaining how they did so. This is a typical situation when using neural networks for deep learning. Beyond that, we can find at least three other obstacles when applying machine learning in finance as described in Israel *et al.* (2020):

Low signal-to-noise ratio Unlike traditional use cases of machine learning techniques such as image processing or natural language processing, the signal-to-noise ratio in financial data is very low. In other words, there is too much noise in the data and therefore the system is not predictable enough. It is worth pointing out that if we consider a classification or regression problem, noise is present not only in the features but also in the labels. For example, we can consider the return of an asset to be the sum of the expected return and the noise. We can easily observe realized returns from market data, but it is very difficult to know what the real expected return is, especially if the noise is very strong.

Market regime switch The market is constantly evolving, and that's why the term "regime switch" is used to describe a change in the interconnections between several components of a financial system. As a classic example, we use expansion or recession to describe different regimes of the economic cycle and bull or bear markets to describe financial market cycles. In order to train a robust machine learning model, in addition to ensuring that the training and validation datasets have a similar distribution, we must ensure that future data also follow the same distribution. Due to the sequential nature of most financial data, these two points are difficult to achieve. Therefore, if we want to have a machine learning model that can be adapted to the current regime, the inputs used to train the model should not include data that is too old.

Not enough data Unlike other industries, the financial sector has fewer big data scenarios. The main issue is that all financial data are collected and are stochastic and non-stationary, making it difficult to produce new data through experimentation that tracks the temporal dynamics of the financial market. In addition, size limitations come not only from features but also often from labels, as we prefer to be able to directly predict returns on financial assets for which there are limited observations. For example, if we have 20-year daily returns on a financial asset, there are only about 5,000 data points, also distributed among different market regimes. For some extreme events, the number of occurrences is much lower, such as the financial crisis, Covid-19, etc. This order of magnitude of data is insufficient to train a deep learning model relative to its number of parameters, especially in the multivariate case. Therefore, in addition to the reasons introduced in the previous paragraph, the volume of the dataset used to train the model is frequently constrained.

4.2 The balance between model complexity, calibration quality and prediction quality

When we use machine learning techniques in quantitative investing, especially when predicting the sign or value of asset returns, we tend to feed all features into a complex deep learning model, hoping to capture the non-linear relationships and complex structures in financial data. Based on the three problems with machine learning in financial applications we've previously mentioned, it's easy for us to make two mistakes:

- Since the quality and quantity of data in finance are not as good as in many other fields where machine learning techniques are used, the accuracy of models will not be high enough to determine whether a model is good and robust. In particular, the labels we use for machine learning, such as the positive and negative signs for asset returns, are very volatile, noisy, and sometimes wrong. Therefore, using these low quality labels and complex models will result in models that learn nothing or overfit.
- In our experience, we have found that the use of complex deep learning models without enough data for training will have a tendency to put more emphasis on certain features, which means that the models tend to overfit. In this case, we will face the problem of losing diversification, which is one of the most important points in quantitative investment.

Thus, in cases where we cannot significantly improve the accuracy of the model, the out-of-sample performance using complex deep learning models with a very large number of features is often inferior to that of traditional rule-based models, because we similarly lose the opportunity to take advantage of diversification when trying to capture more complex relationships in the data. Therefore, we believe we should choose a balance between model complexity and diversification. Inspired by the concept of weak learners in ensemble learning, we tend in practice to train many weak models with simple structures and using fewer features. Then, we build a strong model by weighting the predictions of all models, rather than just training a complex machine learning model. Also, we choose different kinds of weak models as we expect them to obtain useful information from different aspects. When training each kind of weak model, we should use as few parameters/hyperparameters as possible to avoid overfitting, while using different sets of hyperparameters to obtain more robust results. Another advantage of using fewer features to train the model is that the model can be trained well with relatively less data, even with neural network structures. In this way, our strategy remains diverse and is more robust. We believe that this is a very important point when applying machine learning in finance, especially for quantitative investments.

This approach, on top of maintaining the principle of diversification, is consistent with the Vapinik-Chervonenkis theory, which seeks to provide a statistical explanation for the learning process. According to Vapnik (2000), when training a machine learning model, we need to find a balance between model complexity, calibration quality and prediction quality. In the context of deep learning, model complexity refers to the number of parameters used in the model, such as the number of layers of the neural network and the number of nodes per layer. More parameters mean more complex models, which can learn more complex relationships in the data, but requires more data for model calibration in the training process. Calibration quality is measured in terms of training error and the complexity of the model and the data have to match each other to increase the calibration quality. For example, a model that is too simple cannot capture all the information in the data, and a linear model cannot find the nonlinear relationships in the training data set, while a model that is too complex requires a large amount of high-quality data for training, which is usually not possible. Prediction

quality relates to the out-of-sample error of the model, and a complicated model is frequently prone to overfitting, which can result in large out-of-sample errors.

As explained in Section 4.1, building very complex machine learning models with data from the financial domain is extremely difficult because the data used to train the models contains numerous flaws. Therefore, our goal is to sacrifice model complexity to obtain better calibration and prediction quality, which means less difficulty fitting the model with the financial data and less out-of-sample error. We believe this is a very suitable solution for the application of machine learning in quantitative investing, where the quality and quantity of data are relatively lower than in industry and we cannot find a definitive law to describe exhaustively the action in the market. The sacrifice of model complexity will be compensated by the diversification of quantitative investments. Therefore, when we want to apply a traditional machine learning model or a deep learning model to predict the future return of an asset, we prefer to use low-complexity models with simple structures and fewer features, such as a Transformer model, which has little depth in its encoder and decoder parts, and uses two or three features for inputs. In this case, deep learning models are still used in order to take advantage of their ability to handle time series and capture long-term memory in the data, rather than use their ability to fit complex nonlinear relationships with very deep structures.

5 Empirical results

5.1 Trend-following investing

Often considered as one of the most straightforward strategies, time-series trend-following strategies consists of taking long positions on assets with positive excess returns over a specified lookback period and taking short positions on assets as their trend declines with the expectation that the price movement is expected to continue. Typically, we consider using 1-month, 3-month and 12-month look-back periods to capture short-, medium- and long-term trends. The strategy is typically implemented through a set of liquid futures such as commodity futures, equity futures, currency forwards and government bond futures. Among the many research papers on trend-following strategies, Lempérière *et al.* (2014) backtested this strategy over a two-hundred-year time frame and identified patterns that existed across asset classes and various study periods.

In our test, we consider a classic time-series trend-following strategy, as described in Chapter 12 of the book of Pedersen (2015). In his book, the author shows the backtest of a time series trend following strategy from January 1985 to June 2012, using 24 commodity futures, 9 equity index futures, 13 bond futures and 12 currency forwards[8], with the following formula to determine the size of the asset allocation for each rebalancing date t:

$$w_{t+1}^{i,m} = \text{sign}\left(\text{excess return of asset } i \text{ over past } m \text{ months}\right) \frac{\sigma^\star}{\sigma_t^i}$$

[8] **Commodity futures**: Aluminum, Brent Oil, Cattle, Cocoa, Coffee, Copper, Corn, Cotton, Crude Oil, Gas Oil, Gold, Heating Oil, Hogs, Natural Gas, Nickel, Platinum, Silver, Soybeans, Soy Meal, Soy Oil, Sugar, Gasoline, Wheat, Zinc;
Currency forwards: AUD-NZD, AUD-USD, EUR-JPY, EUR-NOK, EUR-SEK, EUR-CHF, EUR-GBP, AUD-JPY, GBP-USD, EUR-USD, USD-CAD, USD-JPY;
Equity index future: Australia, Germany, Spain, France, Italy, Japan, UK, US;
Bond futures: 3 Yr Australian Bond, 10 Yr Australian Bond, 2 Yr Euro - Schatz, 5 Yr Euro - Bobl, 10 Yr Euro - Bund, 30 Yr Euro - Buxl, 10 Yr CGB, 10 Yr JGB, 10 Yr Long Gilt, 2 Yr US Treasury Note, 5 Yr US Treasury Note, 10 Yr US Treasury Note, 30 Yr US Treasury Bond.

where σ_t^i is the ex-ante annualized volatility for each instrument, estimated using an exponentially weighted average and $\sigma^\star$ is a chosen annualized volatility target. This scaling of volatility is designed to normalize the different levels of volatility across assets and time to ensure that each asset has a similar risk contribution to the overall portfolio. Based on the backtest results, the author argued that these single-asset trend-following strategies provide positive performance in almost every case, i.e., a combination of assets and the backward-looking window used to measure the trend, and emphasized that the benefits of diversification are significant due to the low pairwise correlation between these strategies.

We follow the same framework to build a baseline time series trend following strategy as a benchmark and test some machine learning methods to try to improve the strategy. To eliminate the bias caused by commodity price increases since the COVID-19 and the Russia-Ukraine war, our dataset includes five futures contracts on global equity indices such as the S&P 500, Eurostoxx 50, etc., as well as five futures contracts on 10-year sovereign bonds such as those of the US, Germany, Australia, Canada, and the UK. Their descriptive statistics of performance and risk are given in Table 4 in Appendix A.

We consider a classification to predict the sign of each asset's excess return for next week, using the different models, such as logistic regression, support vector machine and the encoder with the attention mechanism. As explained in Section 4.1, for each model, we train weak learners with one or two features and aggregate the results of all weak learners to build a strong learner. To avoid selection bias, we calculate the short-, medium-, long-term trend of the asset with 5 classical lookback windows (1-month, 2-month, 3-month, 6-month, 12-month), using the following formulas:

$$\text{TSMOM}_t^{i,m} = \frac{1}{m} \sum_{s=t-m+1}^{t} \tilde{r}_s^i, \quad \tilde{r}_s^i = \frac{\sigma^\star}{\sigma_s^i} r_s^i$$

where m is the number of days in the lookback window, r_t^i is the return of asset i at t, σ_t^i is the maximum of the estimated annualized volatilities using 5 lookback windows (1-month, 2-month, 3-month, 6-month, 12-month) as the ex-ante volatility and $\sigma^\star$ is the annualized volatility target. This operation is designed to normalize the different levels of volatility of the assets, then we can use an equal weight approach to construct the portfolio:

$$w_{t+1}^{i,m} = \begin{cases} 1 & \text{if } \text{TSMOM}_t^{i,m} > 0 \\ -1 & \text{if } \text{TSMOM}_t^{i,m} < 0 \end{cases}$$

The realized gain of the strategy with a m-day lookback window for $t+1$ is then calculated as:

$$R_{t+1}^m = \frac{1}{N} \sum_{i=1}^{N} w_{t+1}^{i,m} \times \tilde{r}_{t+1}^i$$

where N is the number of assets in the portfolio.

Binary classification In a binary classification problem, we have $(x_1, y_1), \ldots, (x_n, y_n)$ that are n observations of (X, Y), where $x_i = (x_{i1}, \ldots, x_{id}) \in \mathbb{R}^d$ is called the feature and $y_i \in \{0, 1\}$ is called the label. We denote the joint probability distribution of (X, Y) by $\mathbb{P}(X, Y)$ and we assume that X and Y have some linear or nonlinear relationship between them and that we can use a classifier function $f : \mathbb{R}^d \to \{0, 1\}$ to describe this relationship $Y = f(X)$. The goal of using machine learning models is to find a classifier function $\hat{f}$ to approximate f in order to predict Y for a given X, such as

$$Y = \hat{f}(X) + \varepsilon$$

where ε denotes the noise. Training a model means calibrating the function $\hat{f}$ with a dataset of n observations to minimize noise.

To fairly compare the performance of strategies using machine learning models with that of the benchmark strategy, the features used to train the model are the same inputs used by the benchmark strategy, such as the short-, medium-, long-term trend of the asset with 5 classical lookback windows (1-month, 2-month, 3-month, 6-month, 12-month). In this case, we make sure that the machine learning model-based strategy and the benchmark strategy use the same level of information in the data, and we want to know if the machine learning model-based strategies can have some computational advantage, that is, they can capture more useful information in the data. As explained in Section 4.1, we prefer to train a few simple machine learning models for each asset rather than a complex cross-asset model with all assets at once. Thus, for each asset i, we have $x_t^i = \left(\text{TSMOM}_t^{i,1m}, \text{TSMOM}_t^{i,2m}, \text{TSMOM}_t^{i,3m}, \text{TSMOM}_t^{i,6m}, \text{TSMOM}_t^{i,12m}\right)$ as the feature to train the model.

As for the labels to be used in the model, we choose to predict the sign of the next week's excess return for each asset, in order to strike a balance between the number and quality of the labels. The daily returns are too volatile and noisy to predict, while forecasting the monthly returns would significantly reduce the size of the training set. In this research, we compare the results of several models, including Logistic regression, SVM, and Encoder with the multi-head attention mechanism. For the portfolio construction, we show in this paper the result of using an equal weight approach as used in benchmark strategy:

$$w_{t+1}^{i,m} = \begin{cases} 1 & \text{if } \hat{f}\left(x_t^i\right) > 0 \\ -1 & \text{if } \hat{f}\left(x_t^i\right) < 0 \end{cases}$$

where $\hat{f}$ denotes the machine learning model to test in this paper, such as $\hat{f}_{\text{LR}}$, $\hat{f}_{\text{SVM}}$, $\hat{f}_{\text{Encoder}}$.

The purpose of this backtest is to illustrate the importance of the trade-off between model complexity and diversification when applying machine learning models to quantitative investing and to test if machine learning models could capture more information in the data. Different models are used to capture the information from different aspects of the data:

- The baseline strategy is equivalent to a linear classifier trained on only one feature, e.g., 12-month momentum, with the threshold set to 0[9].
- The logistic regression is also to build a linear classifier trained on only one feature, but the thresholds are learned from the data.
- To capture the nonlinear relationships in the data using the RBF kernel (Radial basis function kernel) used in SVM, we train a model using two features at a time, say 1-month and 12-month momentum. We believe that it is better to train many weak learners using one combination of features at once than to train one strong learner directly with all features together, which means that we have to train $C_5^2 = 10$ models when we have 5 features.
- We also trained the encoder with the multi-head attention mechanism to capture information in the time series dimension. The features used to calibrate the models are still trends using 5 different lookback windows, but these features are fed into the

[9]When the 12-month momenum is greater than 0, we will go long the asset and when it is below 0, we will go short the asset.

models as time series. We will apply the same idea as before, training many weak learners to build a strong learner, so each model is also trained with two features, even though in our case we are using a deep learning model.

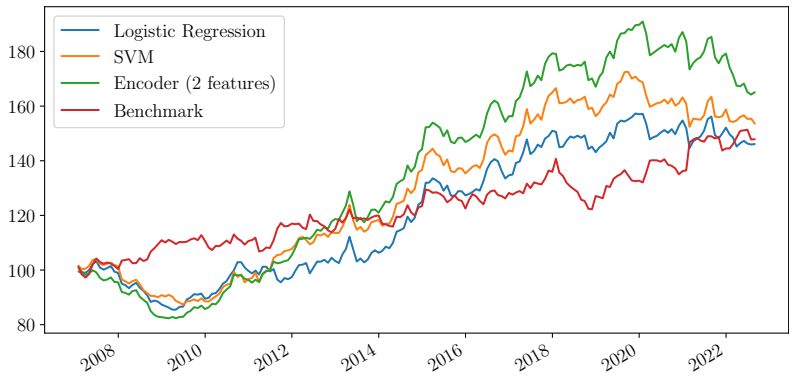
For each case, if we need to choose hyperparameters for the model, we will choose a set of hyperparameters instead of one and average the prediction results for all models using different hyperparameters. For example, we need to choose a hyperparameter C for the SVM model, which serves as the importance of misclassification and the choice of hyperparameters is always abstract and often leads to selection bias. That is why we choose different values of C in our case, such as $C = 0.5, 1, 2$, in order to obtain a robust result.

For our encoder with multi-head attention mechanism, we used a simple structure of 2 encoder blocks, which means that $N = 2$ as shown in Figure 12. Since our design is to train models using only a small number of features as input, this linear layer does not need to have too many nodes. In our case, the number of nodes in the linear layer is set to 8. Then, for the same reason, the number of heads is set to 4 and the number of node in feed forward network is also set to 4. To accelerate the training process, we used an early stopping method.

For the backtesting framework setup, we use a weekly rebalance and the transaction cost on the futures was set to 2bps. We recalibrate our machine learning models every 3 months with observed data from the last 3 years. Due to random effects in the model training process, we trained several models on the same training dataset at each recalibration date and then averaged the prediction results of all models. As explained and emphasized throughout this paper, our goal is to train many weak learners and the structure of each model is simple; the risk of overfitting for each model is relatively low. This is an important advantage when we try to apply machine learning in finance, because there is a notion of time in financial data, and very often the data in the training set and the data in the test set do not follow the same distribution, which leads to huge out-of-sample errors by using too complex models.

Backtest results In Figure 13, we have reported the cumulative performance of the benchmark strategy and all strategies that use the machine learning model. Moreover, the descriptive statistics of performance and risk are given in Table 1.

Figure 13: Cumulative performance of all strategies



Source: Authors' calculations.

Table 1: Performance of different strategies

	μ	σ	SR	MDD	ξ
Logistic Regression	2.37%	6.25%	0.38	-17.23%	2.76
SVM	2.68%	6.22%	0.43	-15.98%	2.57
Encoder (2 features)	3.30%	6.30%	0.52	-17.59%	2.79
Benchmark	2.47%	5.72%	0.43	-13.12%	2.29

Source: Authors' calculations.

μ is the annualized return, σ is the annualized volatility, SR is the Sharpe ratio[10], MDD is the maximum drawdown and ξ is the skew measure, which is the ratio between the maximum drawdown and the volatility[11].

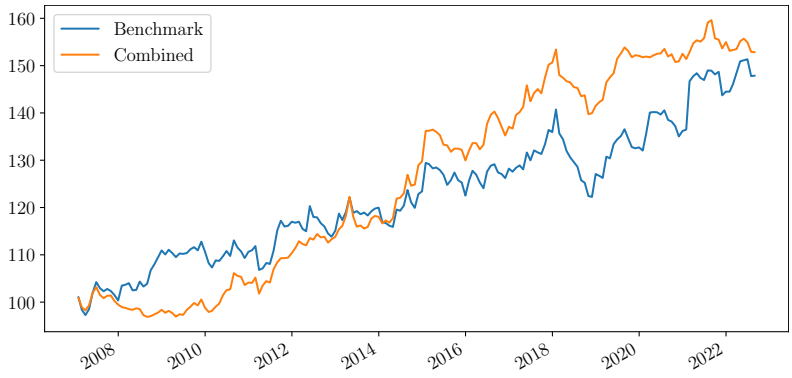
From Figure 13 and Table 1, we notice that all strategies using machine learning models perform very differently relative to the benchmark strategy. Logistic regression and SVM models have Sharpe ratios comparable to the benchmark strategy, and the encoder with the multi-head attention mechanism outperforms all other models. All strategies have the same level of volatility and maximum drawdown, especially the skew which is the ratio of the maximum drawdown to the volatility. Our three machine learning models have different levels of model complexity and use different numbers of features to train. In this regard, the more complex the model, the more likely it is to capture the useful information in the data. However, as we have discussed in Section 4.1, the evolution of financial markets is continuous, and machine learning models trained on historical data may not be applicable in the current environment. We have observed this in our backtest: all machine learning models struggled after 2020, a period in which Covid-19, high inflation, interest rate hikes, high correlations between stocks and bonds, and the Russian-Ukrainian war made financial markets different than ever before. The more complex the model, the worse our performance will be in this period because of the large out-of-sample error. This is why the encoder model declines rapidly after 2021. In this case, the viable option to increase the frequency of model recalibration and use more recent data in the training process in order to quickly adapt to actual market conditions. As the backtesting results show, using machine learning models can capture more useful information in the data to predict future returns, so we can try to improve the strategy by adding some macro factors to the model.

All these models are designed to capture information about different dimensions of the data, single feature or double feature, linear or non-linear, time series or static, so we can aggregate all these models to achieve the advantage of diversification. For this reason, and to maintain maximum generality, we consider a risk parity approach for portfolio construction. However, as we saw in the previous section, all our machine learning models are trained within a rolling 3-year window for each recalibration date. Therefore, all these models inevitably have some similarities. Thus, we can try to first apply a risk parity approach to all machine learning strategies to get a new strategy, and then apply another risk parity approach or an equal-weighted approach to this new strategy and the benchmark strategy.

In Figure 14 and Table 2, we show a comparison between the benchmark strategy and the new strategy, which combines three machine learning strategies and the benchmark strategy with a two-level risk parity approach. The combined strategy outperformed the benchmark strategy with a higher Sharpe ratio and a much lower maximum drawdown. This result suggests that machine learning models may offer some computational advantages in quantitative investing. In particular, it confirms the key idea of our strategy design, which

[10]For the sake of simplicity, the risk-free rate is set to 0.
[11]If ξ is greater than 3, this indicates that the strategy has a high skewness risk.

Figure 14: Cumulative performance of strategies



Source: Authors' calculations.

Table 2: Performance of different portfolio construction

	μ	σ	SR	MDD	ξ
Benchmark	2.47%	5.72%	0.43	-13.12%	2.29
Combined	2.70%	4.45%	0.61	-8.92%	2.00

Source: Authors' calculations.

is to train many weak machine learning models to build a strong learner, to ensure a balance between model complexity and maintaining diversification.

5.2 Multi-Period Portfolio Optimization

A natural development of Harry Markowitz's mean-variance optimization (MVO) model is multi-period portfolio optimization. The objective is to identify the dynamic asset allocation strategy by taking into account inter-temporal effects such as rebalancing costs, trading impacts, time-varying constraints, price trends, etc. In our previous paper (Lezmi *et al.*, 2022), we proposed several efficient numerical algorithms to solve the multi-period portfolio alignment problem defined by Le Guenedal and Roncalli (2022, pages 36-37), including augmented quadratic programming, cyclical coordinate descent and alternating direction method of multipliers. These advanced algorithms are explained in great detail in the survey of Perrin and Roncalli (2020) in the context of portfolio optimization, e.g., risk parity portfolios, strategic asset allocation, smart beta portfolios, minimum-variance strategies, regularized allocation problems and turnover management.

Let us consider a universe of n assets. We define the following h-period optimization problem:

$$\begin{aligned} x^\star &= \arg \max_{x_{t+1},\, x_{t+2},\ldots} \mathbb{E}\left[\mathcal{U}\left(x_{t+1},\ldots,x_{t+h}\right) \mid \mathcal{F}_t\right] \\ &\text{s.t.} \quad x \in \Omega \end{aligned} \tag{6}$$

where $x_t = \left(x_{1,t},\ldots,x_{n,t}\right)$ is the vector of the portfolio weights at the t^{th} period, $x = (x_{t+1}, x_{t+2},\ldots,x_{t+h})$ is the vector of the h allocations, $\mathcal{U}\left(x_{t+1},\ldots,x_{t+h}\right)$ is the inter-temporal utility function, $\mathcal{F}_t$ is the filtration associated with the probability space, and $x \in \Omega$ is a set of linear and non-linear constraints. As explained in Lezmi *et al.* (2022), we

are only interested in $x^\star_{t+1}$ in the solution $x^\star = \left(x^\star_{t+1}, x^\star_{t+2}, \ldots, x^\star_{t+h}\right)$ of Problem (6). Since the filtration at time $t+1$ will be updated, the optimal solution $x^\star_{t+2}$ founded at time t is no longer valid. The right formulation of Problem (6) is then:

$$\begin{aligned} x^\star_{t+1} &= \arg\max_x f(x) \\ &\text{s.t.} \quad x \in \Omega \end{aligned} \tag{7}$$

In particular, we consider a mean-variance optimization with ridge penalization in our experiment to illustrate the application of Transformer models. Therefore, Problem (6) becomes:

$$\begin{aligned} x^\star_{t+1} &= \arg\min_x \sum_{s=t+1}^{t+h} \left\{ \frac{1}{2} x_s^\top \Sigma_s x_s - \gamma x_s^\top \mu_s \right\} + \frac{\lambda_s}{2} \left\| x_s - x_{s-1} \right\|_2^2 \\ &\text{s.t.} \quad x \in \Omega \end{aligned} \tag{8}$$

where x_t is the current portfolio weight, γ is a coefficient that controls the trade-off between the portfolio's expected return and its volatility and λ_s is the penalty coefficient for turnover.

However, multi-period portfolio optimization models are rarely used in practice. One reason is that it can be quite challenging to accurately estimate the return/risk for multiple periods or even just one period. In the framework of the MVO model, we need to estimate the expected return vector μ and variance-covariance (VCV) matrix Σ of the assets in a portfolio. In addition, the VCV matrix can be split into a volatility vector and a correlation matrix. Empirically, the expected return vector is considered the most difficult to estimate of these three inputs to the MVO model, while the correlation matrix is typically regarded as being more stable than the expected return and volatility. Therefore, volatility forecasting is an important issue in quantitative research.

As a standard measure of market risk, volatility is widely used in a variety of applications throughout the financial industry. In particular, all traditional portfolio construction methods take the volatility of assets as an input to the model, not only in the mean-variance optimization approach but also in the risk parity/risk budgeting approach. Both methods are described in Appendix B.6 and B.7. At its core, the volatility forecasting problem can be viewed as a time series forecasting problem. For a long time, GARCH model and its extensions are the state-of-the-art models. As explained in Section 2.2, we can use these models iteratively or directly to perform multi-step forecasting. However, both methods have their shortcomings: the iterative method accumulates prediction errors, while the direct method fails to model the relationship between predictions. In our study, we will leverage the attention mechanism and the seq2seq architecture in the Transformer model to address these issues. As we have explained in Section 3, Transformer models are suitable for modeling complex relationships in sequence data and for multi-step time series forecasting.

In this section, we consider a classical small investment universe and our dataset consists of 2 futures contracts on equity index S&P 500 and Eurostoxx 50 and 2 futures contracts on 10Y sovereign bonds of the US and Germany. In our experiments, instead of directly forecasting the entire VCV matrix using the Transformer model, we forecast the volatility of each asset separately and then build the VCV matrix using the correlation matrix estimated from the 1-year historical asset returns. The reason for this is that the Transformer model is not guaranteed to return a positive semi-definite matrix and multivariate time series forecasting needs more complex structure in the model and more data to train the model. Therefore, it's often more difficult to obtain good results. The phenomenon known as the "curse of dimensionality" was first introduced by Richard E. Bellman (Bellman, 1961), when

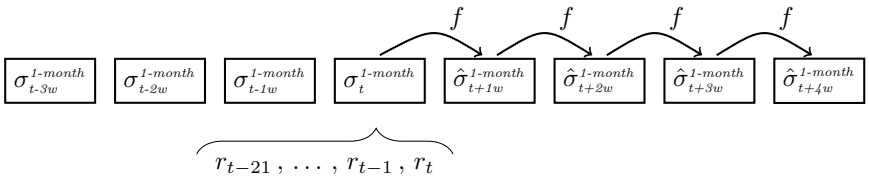
he considered the problem of dynamic programming. In addition, the correlation relationship is generally considered to be more stable relative to return and volatility. Therefore, we use the historical correlation matrix and consider that the correlation matrix will not change during multi-step forecasting.

More precisely, the input to the model is the 1-month rolling volatility time series of assets and we train a Transformer model for each asset. We train the model with weekly data from 2000 and 2018 and we split the data into training and validation subsets, representing 75% and 25% of all samples, respectively. We consider rebalancing our portfolio once a month. Therefore, our objective is to train a forecasting model to predict the volatility of assets in the coming month at each rebalancing date. One of the benefits of the Transformer model, as we discussed in Section 3.3, is that it can perform multi-step forecasting. For example, let $\sigma_t^{1\text{-}month} = \sqrt{\frac{\sum_{i=t}^{t-21}(r_i-\bar{r})^2}{21-1}}$ denote 1-month realized volatility at date t, where $\bar{r} = \frac{\sum_{i=t}^{t-21} r_i}{21-1}$. In the case of forecasting the time series of 1-month rolling volatility $\{\sigma_t^{1\text{-}month}\}_{t\in\mathcal{T}}$, we can train a Transformer model with weekly data $\{\ldots, \sigma_{t-2w}^{1\text{-}month}, \sigma_{t-1w}^{1\text{-}month}, \sigma_t^{1\text{-}month}\}$ and predict the next values 4 times, as shown in Figure 15. Therefore, $\hat{\sigma}_{t+4w}^{1\text{-}month}$ is the forecast of volatility for the next one month at date t and using the correlation matrix estimated from historical data, we can build a forward-looking VCV matrix, which will be introduced as input in different portfolio construction methods. Training the model with weekly data is a trade-off between the amount of data, the overlap of the data and the number of prediction steps:

- If we train the model with daily data, we will have more data to train the model, but in forecasting we can only forecast one day at a time. In a typical case, we would choose to rebalance the portfolio at a monthly frequency, which requires estimating the forward-looking VCV matrix for the next one month. In this case, we would need to forecast for 21 consecutive steps and the accumulated error would be huge.

- If we train the model with monthly data, we only need to predict the next one step immediately after to estimate the future 1-month volatility. However, the amount of data would be greatly reduced and there would not be enough data to train a deep learning model.

Figure 15: Using Transformer models to forecast the time series of 1-month rolling volatility



Source: Amundi Institute.

In our experiment, we consider three different portfolio allocation methods:

- Single-period MVO portfolio with monthly rebalancing

- Risk parity portfolio with monthly rebalancing

- Multi-period MVO portfolio with weekly rebalancing as described by Problem (8)

For the sake of simplicity, we will consider the case where there are only *no cash and leverage* and *no short selling* constraints. Therefore, Problem (8) becomes:

$$x_{t+1}^{\star} = \arg\min_{x} \sum_{s=t+1}^{t+h} \left\{ \frac{1}{2} x_s^{\top} \hat{\Sigma}_s x_s - \gamma x_s^{\top} \mu \right\} + \frac{\lambda_s}{2} \left\| x_s - x_{s-1} \right\|_2^2 \tag{9}$$

$$\text{s.t.} \quad \begin{cases} \mathbf{1}_n^{\top} x_s = 1 \\ \mathbf{0}_n \leq x_s \leq \mathbf{1}_n \end{cases} \qquad \text{for } s = t+1, \ldots, t+h$$

In general, we use the historical asset returns and VCV matrix for the past 12 months as input to these portfolio allocation problems. In our case, we train a Transformer model for each asset to forecast the volatility for the next 4 time steps (weeks) and the last one will be the estimation of the future 1-month volatility. We then create a VCV matrix based on this prediction and the historical correlation matrix, which will be the input to the portfolio allocation problem described above. We compare the portfolio using empirical estimation with the portfolio that uses Transformer models for multi-step forecasting.

We show the backtest results in Figures 16, 17 and 18. Given that our testing period coincides with an economically challenging period caused by Covid-19 and the Russia-Ukraine War, and that all three portfolios are long-only portfolios, they all experienced significant losses in 2022. However, the performance of portfolios that use Transformer models' forecasts as input is superior to that of portfolios using historical estimates. We show descriptive statistics of all backtest in Table 3. We notice that the portfolios using Transformer models have a higher Sharpe ratio. Our weekly rebalanced multi-period MVO portfolios performed better than the single-period MVO portfolios. As Risk Parity portfolios only use estimates of the VCV matrix as input, we do not include errors in the estimation of returns in the model, which are typically more difficult to estimate relative to the VCV matrix. As a result, Risk Parity portfolios outperformed MVO portfolios in this economically challenging period.

Figure 16: Single-period MVO portfolio

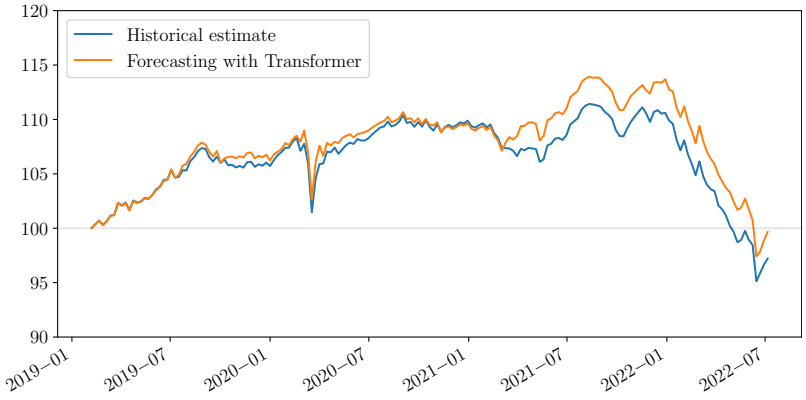


Figure 17: Risk Parity portfolio

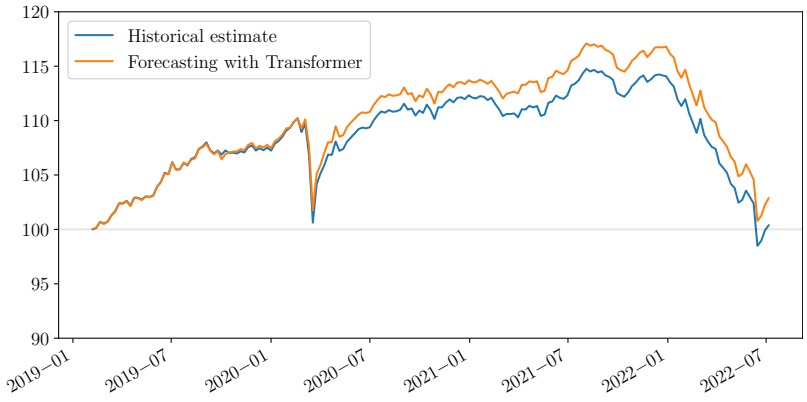
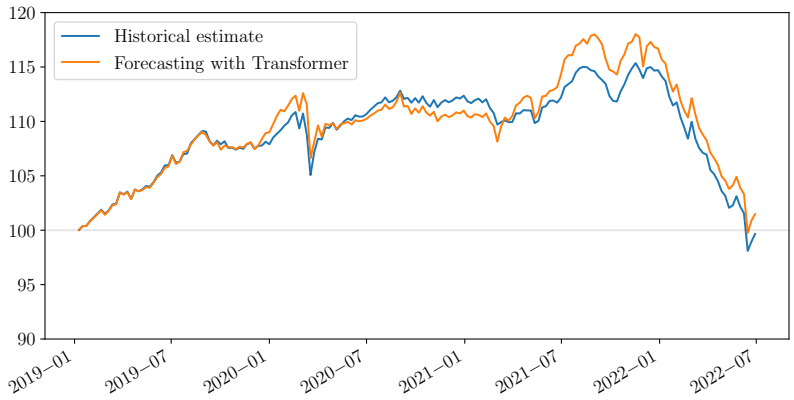


Figure 18: Multi-period MVO portfolio



Source: Authors' calculations.

Table 3: Descriptive statistics of portfolios using different portfolio allocation approaches

		μ	σ	SR	MDD	ξ
Single-period MVO	Historical estimate	-0.99%	5.15%	-0.19	-14.63%	2.84
	Forecasting with Transformer	-0.33%	5.27%	-0.06	-14.49%	2.75
Risk Parity	Historical estimate	-0.02%	5.72%	0.00	-14.18%	2.48
	Forecasting with Transformer	0.66%	5.55%	0.12	-13.93%	2.51
Multi-period MVO	Historical estimate	-0.44%	4.91%	-0.09	-14.96%	3.05
	Forecasting with Transformer	0.11%	5.61%	0.02	-15.48%	2.76

Source: Authors' calculations.

6 Conclusion and Future research

In this paper, we detail the attention mechanism and the seq2seq architecture used by Transformer models, two features that allow them to handle sequence learning problems well. We explore two applications of Transformer models to time series forecasting in portfolio management: using only the encoder part of the Transformer model to build a trend-following strategy and using both the encoder and decoder parts to forecast volatility in a multi-period portfolio optimization problem. Our results show that the attention mechanism can capture complex relationships in sequential data and that the Transformer model is a powerful tool for multi-step time-series forecasting, which is a difficult task for traditional statistical models, such as ARIMA and GARCH.

We also discuss the challenges of applying machine learning to finance. We believe that we must confront the differences between finance and other areas where machine learning techniques have been successfully applied. Since most machine learning models learn the relationship between inputs and outputs in a completely data-driven way, we need to maintain a balance between model complexity, calibration quality and prediction quality. In particular, when we apply machine learning techniques to build quantitative investment strategies, it's crucial to make sure the portfolio is well diversified. Otherwise, we will easily fall into the trap of overfitting and selection bias, which will result in significant differences between our in-sample and out-of-sample results. As a result, applying machine learning to finance is challenging and requires sufficient economic/financial knowledge to design a robust framework.

There are at least two ways to improve our work. First, as we describe in this paper, the main difficulty when applying machine learning techniques in finance is that the signal-to-noise ratio in financial data is often weak. Therefore, the next phase of research focuses on financial data denoising and labeling, which is the key to the successful application of machine learning techniques in finance. Feature engineering is also important for time series forecasting, where we can decompose time series into trend, seasonality and noise components through pattern decomposition techniques. Using these methods in conjunction with deep learning models is an interesting research topic. Second, as we described in our working paper about graph neural networks (Pacreau *et al.*, 2021), the attention mechanism is also used in the graph attention layer (GAT) to capture underlying relationships among data dimensions. Therefore, it is natural to try to combine Transformer models and graph neural networks (GNNs) to manage multivariate and spatio-temporal time series data, such as traffic forecasting. Some researchers have claimed that this combination of models can improve performance and provide a better understanding of the causality of data during spatio-temporal time series forecasting, such as Cai *et al.* (2020) and Xu *et al.* (2020). In the field of finance, the correlation or supply chain relationship between multiple companies can be regarded as a spatial relationship. Therefore, combining Transformers and GNNs to model the dynamics of time series and the dependency among dimensions is an important future avenue for our research. This will open the door to a new field of research into capturing more complex relationships in financial data and improving quantitative investment strategies.

References

Bellman, R.E. (1961), *Adaptive Control Processes*, Princeton University Press.

Cai, L., Janowicz, K., Mai, G., Yan, B. and Zhu, R. (2020), Traffic Transformer: Capturing the Continuity and Periodicity of Time Series for Traffic Forecasting, *Transactions in GIS*, 24(3), pp. 736-755.

Chaves, D.B., Hsu, J.C. and Shakernia, O. (2012), Efficient Algorithms for Computing Risk Parity Portfolio Weights, *Journal of Investing*, 21(3), pp. 150-163.

Cherief, A., Ben Slimane M., Dumas J.-M. and Fredj, H. (2022), Credit Factor Investing with Machine Learning Techniques, *SSRN*, https://ssrn.com/abstract=4155247.

Feng, Y. and Palomar, D.P. (2015), SCRIP: Successive Convex Optimization Methods for Risk Parity Portfolio Design, *IEEE Transactions on Signal Processing*, 63(19), pp. 5285-5300.

Gehring, J., Auli M., Grangier, D., Yarats, D. and Dauphin, Y.N. (2017), Convolutional Sequence to Sequence Learning, *Proceedings of the 34th International Conference on Machine Learning*, PMLR 70, pp. 1243-1252.

Goodfellow, I., Bengio Y. and Courville, A. (2016), *Deep Learning*, The MIT Press.

Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H. and Schmidhuber, J. (2009), A Novel Connectionist System for Improved Unconstrained Handwriting Recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5), pp. 855-868.

Griveau-Billion, T., Richard J.C. and Roncalli, T. (2013), A Fast Algorithm for Computing High-dimensional Risk Parity Portfolios, *SSRN*, https://ssrn.com/abstract=2325255.

Grover, A. and Leskovec, J. (2016), node2vec: Scalable Feature Learning for Networks, *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 855-864.

Hastie, T., Tibshirani R. and Friedman, J. (2009), *The Elements of Statistical Learning*, Second edition, Springer.

He, K., Zhang, X., Ren, S. and Sun, J. (2016), Deep Residual Learning for Image Recognition, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778.

Hochreiter, S., and Schmidhuber, J. (1997), Long Short-Term Memory, *Neural Computation*, 9(8), pp. 1735-1780.

Hubert, T., Schrittwieser, J., Antonoglou, I., Barekatain, M., Schmitt, S. and Silver, D. (2021), Learning and Planning in Complex Action Spaces, *In Proceedings of the 38th International Conference on Machine Learning, PMLR 139.*

Israel, R., Kelly, B. and Moskowitza, T. (2020), Can Machines Learn Finance?, *Journal of Investment Management*, 18(2), pp. 23-36.

Jagannathan, R., and Ma, T. (2003), Risk Reduction in Large Portfolios: Why Imposing the Wrong Constraints Helps, *Journal of Finance*, 58(4), pp. 1651-1684.

Kazemi, S.M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., Wu, S., Smyth, C., Poupart, P. and Brubaker, M. (2019), Time2Vec: Learning a Vector Representation of Time, *arXiv preprint*, arXiv:1907.05321.

Le Guenedal, T. and Roncalli, T. (2022), Portfolio Construction with Climate Risk Measures, *Climate Investing: New Strategies and Implementation Challenges*, edited by Emmanuel Jurczenko, Wiley, pp. 49-86.

Lempérière, Y., Deremble, C., Seager, P., Potters, M., Bouchaud, J.P. (2014), Two centuries of trend following, *Journal of Investment Strategies*, 3(3), pp. 41-61.

Lezmi, E., Roncalli, T. and Xu, J. (2022), Multi-Period Portfolio Optimization and Application to Portfolio Decarbonization, *SSRN*, https://ssrn.com/abstract=4078043.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y. and Yan, X. (2019), Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting, In *NeurIPS 2019*.

Markowitz, H. (1952), Portfolio Selection, *Journal of Finance*, 7(1), pp. 77-91.

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013), Efficient Estimation of Word Representations in Vector Space, *arXiv preprint*, arXiv:1301.3781.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y. and Jaiswal, S. (2017), graph2vec: Learning Distributed Representations of Graphs, *arXiv preprint*, arXiv:1707.05005.

Pacreau, G., Lezmi, E. and Xu, J. (2021), Graph Neural Networks for Asset Management, *ResearchGate*, https://www.researchgate.net/publication/356634779.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013), On the difficulty of training Recurrent Neural Networks, *ICML'13: Proceedings of the 30th International Conference on International Conference on Machine Learning*, vol. 28, pp. 1310-1318.

Pedersen, L.H. (2015), Chapter 12 Managed Futures: Trend-Following Investing, *Efficiently Inefficient: How Smart Money Invests and Market Prices Are Determined*, Princeton University Press, pp. 208-230.

Perrin, S. and Roncalli, T. (2020), Machine Learning Optimization Algorithms & Portfolio Allocation, in Jurczenko, E. (Ed.), *Machine Learning for Asset Management: New Developments and Financial Applications*, Chapter 8, Wiley, pp. 261-328.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014), DeepWalk: Online Learning of Social Representations, *arXiv preprint*, arXiv:1403.6652..

Roncalli, T. (2013), *Introduction to Risk Parity and Budgeting*, Chapman & Hall / CRC Press.

Tsai, Y.H., Bai, S., Yamada, M., Morency, L.P. and Salakhutdinov, R. (2019), Transformer Dissection: An Unified Understanding for Transformer's Attention via the Lens of Kernel, *In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4344-4353.

Vapnik, V. (2000), *The Structure of Statistical Learning Theory*, Second edition, Springer.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I. (2017), Attention is all you need, *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*.

Xu, M., Dai, W., Liu, C., Gao, X., Lin, W., Qi, G. and Xiong, H. (2020), Spatial-Temporal Transformer Networks for Traffic Flow Forecasting, *arXiv preprint*, arXiv:2001.02908.

Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J. and Sun, L. (2022), Transformers in Time Series: A Survey, *arXiv preprint*, arXiv:2202.07125.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H. and Zhang, W. (2021), Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting, In *AAAI 2021*.

A Data

Table 4: Descriptive statistics of equity index futures and 10Y bond futures from 2000 to 2022

	μ	σ	SR	MDD	ξ
Europe equity index	0.55%	18.37%	0.03	-63.91%	3.48
Hong Kong equity index	2.45%	20.32%	0.12	-58.25%	2.87
Japan equity index	2.94%	19.17%	0.15	-60.98%	3.18
UK equity index	1.39%	13.84%	0.10	-49.56%	3.58
US equity index	4.29%	15.41%	0.28	-58.33%	3.79
Australia 10Y	1.76%	6.35%	0.28	-18.24%	2.87
Cananda 10Y	2.77%	5.49%	0.51	-16.69%	3.04
Germany 10Y	3.33%	5.42%	0.62	-15.12%	2.79
UK 10Y	2.44%	6.06%	0.40	-18.96%	3.13
US 10Y	3.34%	5.73%	0.58	-14.48%	2.53

Source: Authors' calculations.

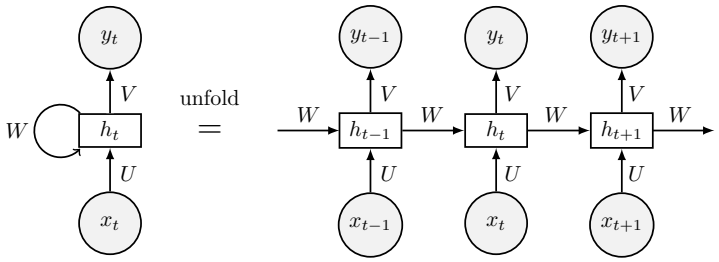
where μ is the annualized return, σ is the annualized volatility, SR is the Sharpe ratio, MDD is the maximum drawdown and ξ is the skew measure, which is the ratio between the maximum drawdown and the volatility.

B Technical appendix

B.1 Recurrent Neural Network

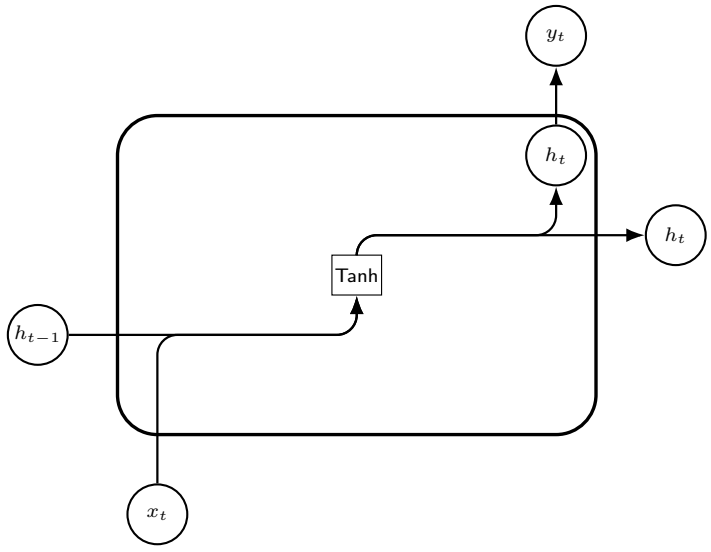
A recurrent neural network is a class of artificial neural networks with recurrent connections. This type of neural network has a feedback loop in its structure that allows it to process sequential data, such as time series, speech and language. For instance, handwriting recognition was the first research result that successfully uses RNNs (Graves *et al.*, 2009). A simple recurrent neural network is composed of an input x_t, a hidden state h_t and an output y_t as shown in Figure 19.

Figure 19: Structure of the RNN model



Source: Amundi Institute.

Figure 20: A typical RNN unit



Source: Amundi Institute.

A typical RNN unit uses a hyperbolic tangent function as the activation function for the hidden state h_t as shown in Figure 20. More precisely, the RNN unit is governed by the

following equations:

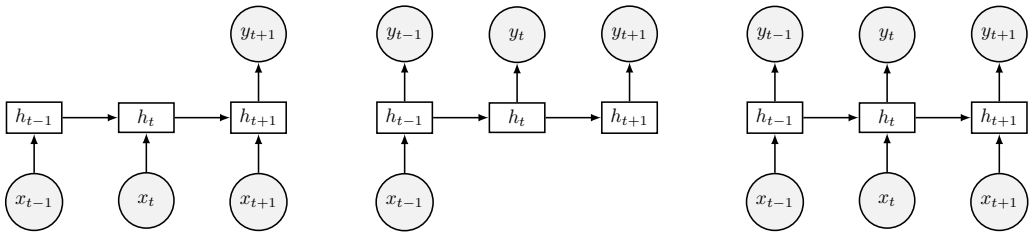
$$\begin{aligned} i_t &= Ux_t + Wh_{t-1} + b_i \\ h_t &= \tanh(i_t) \\ y_t &= Vh_t + b_o \end{aligned} \tag{10}$$

where U, V, W, b_i, b_o are common parameter matrices and vectors shared across time t[12].

The hidden state h_t acts as the "memory" of the network and depends on the current input x_t and the hidden state at the previous time step h_{t-1}. Similarly, the previous hidden state h_{t-1} depends on x_{t-1} and h_{t-2}, and so on. As a result, the hidden state h_t has indirect access to all previous inputs $x_t, x_{t-1}, x_{t-2}, \ldots$. It is in this way that RNNs retain the memory of the past in the data, which allows them to predict very accurately what will happen next.

Different types of RNNs

Figure 21: Different types of RNNs



Source: Amundi Institute.

Usually, we choose a many-to-one model or a many-to-many model to do time series forecasting. It is worth noting that the many-to-one model can also be used for multi-step prediction in the two following ways:

- We can set y_t as a vector containing all of the time steps to be predicted.
- During the inference, we can predict one next step y_t and use them as the new input to recursively predict other time steps $y_{t+1}, y_{t+2}, \ldots$. To do this, y_t should contain all features of x_t, so that we can generate $x_{t+1}, x_{t+2}, \ldots$ as next inputs. This approach leads to the accumulation of errors in the prediction.

Training process of RNNs

The goal of the learning process is to find the best weight matrices U, V and W that give the best prediction of $\hat{y}_t$, starting from the input $x_t, x_{t-1}, x_{t-2}, \ldots$, of the real value y_t. Like in a traditional Feedforward neural network, the loss function is minimized using these two major steps: forward propagation and backward propagation through time. These steps are iterated many times, and the number of iterations is called the epoch number.

[12]This means that once the RNNs are trained, these parameters are fixed for each time step during the inference process.

The vanishing/exploding gradient problem

RNNs will multiply the parameters in the hidden layers during training by the backward propagation through time (BPTT) algorithm. If the gradient is too huge, the gradient explodes and becomes NaN; if the gradient is small, the gradient disappears so that the model stops learning or takes an excessive amount of time to train, both of which have poor model performance.

As shown in Equation (10), the hidden states and outputs at each time steps can be rewritten as:

$$\begin{aligned} h_t &= f\left(x_t, h_{t-1}, U, W, b_i\right) \\ o_t &= g\left(h_t, V, b_o\right) \end{aligned}$$

When training the RNN, we need to apply the gradient descent method. Let us denote L the loss function of the RNN model, then we have:

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

where L_t is the loss at the time-step t such as $L = \sum_{t=1}^{T} L_t$. According the Backpropagation Through Time (BPTT) algorithm,

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

where

$$\begin{aligned} \frac{\partial L_t}{\partial W} &= \sum_{k=1}^{t} \frac{\partial L_t}{\partial i_k} \frac{\partial i_k}{\partial W} \\ \frac{\partial L_t}{\partial i_k} = \delta_{t,k} &= \frac{\partial L_t}{\partial i_{k+1}} \frac{\partial i_{k+1}}{\partial h_k} \frac{\partial h_k}{\partial i_k} \\ &= \delta t_{t,k+1} W \operatorname{diag}\left(f'\left(i_k\right)\right) \end{aligned}$$

Thus,

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \sum_{k=1}^{t} \delta_{t,k} h_{k-1}^{\top}$$

As

$$\begin{aligned} \delta t_{t,k} &= \delta t_{t,k+1} W^{\top} \operatorname{diag}\left(f'\left(i_k\right)\right) \\ &= \delta t_{t,k+2} W^{\top} \operatorname{diag}\left(f'\left(i_{k+1}\right)\right) W^{\top} \operatorname{diag}\left(f'\left(i_k\right)\right) \\ &= \delta_{t,t} \prod_{\tau=k}^{t-1} W^{\top} \operatorname{diag}\left(f'\left(i_\tau\right)\right) \end{aligned} \tag{11}$$

According to Pascanu *et al.* (2013), Equation (11) takes the form of a product of $t-k$ matrices which may shrink to zero or explode to infinity. So, this will lead to the vanishing gradient problem or the exploding gradient problem.

Remark 1. *Why do we use "tanh" as the activation function in RNNs?*

Among the three commonly used activation functions ReLU, sigmoid, tanh, we choose tanh as the activation function for RNNs units. ReLU may have very large outputs, so they may cause the exploding gradient problems. Although both sigmoid and tanh may cause the vanishing gradient problem, the case of sigmoid is much more severe than tanh's. In addition, the gradient calculation cost of tanh is much lower:

$$\tanh'(x) = 1 - \tanh^2(x)$$

The advantages and disadvantages of RNNs

As mentioned in the previous section, RNNs have the following advantages compared to traditional statistical models for time series forecasting:

- Using neural networks, RNNs can find complex patterns in the input time series.
- Using recurrent connections, RNNs can model sequential data so that each sample can be assumed to be dependent on previous ones.
- Comparing RNNs with traditional statistical models, they may have better results in multi-step time series forecasting.

The main issue for RNNs' training is the vanishing gradient or exploding gradient problem when training on long time series, which means that the parameters in the hidden layers either do not change much or they lead to numeric instability. This happens because the gradient of the cost function includes the power of W, which may shrink to zero or explode to infinity. Another main issue of RNNs is that they can not remember long memory. Since the inputs x_t are processed recursively by RNNs, x_{t-m} has little effect on y_t when m is large. Therefore, the memory of RNNs is weak and cannot take into account the elements of the long past. However, it's very crucial in finance to memorize as much history as possible in order to predict the next steps. In addition, the computational cost of RNNs is very high because the training process cannot be parallelized. In summary, all these three problems indicate that RNNs are weak at handling long time sequences. To address this, Long Short-Term Memory networks were proposed by Hochreiter and Schmidhuber (1997).

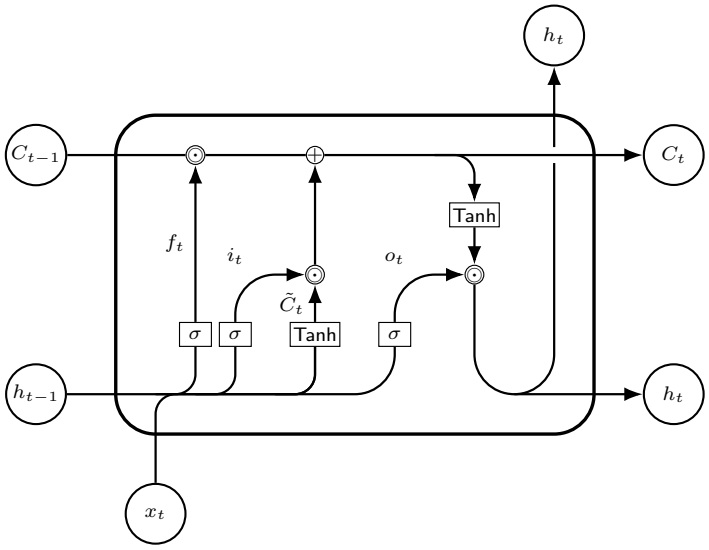
B.2 Long Short Term Memory

Long Short-Term Memory (LSTM) network is an extension of Recurrent Neural Networks (RNNs) proposed by Hochreiter and Schmidhuber (1997). This network is intended to address the problem that RNNs have in dealing with long temporal dependencies. A common LSTM unit has three inputs: the new data x_t, the hidden state of the cell at the previous time step h_{t-1} and the cell state at the previous time step C_{t-1} and is composed of a cell and three gates: a forget gate f_t, an input gate i_t and an output gate o_t as shown in Figure 22. The three gates regulate the flow of information into and out of the cell.

The cell in LSTM units keeps information from its precious state in a hidden vector, which will be combined with the new data in order to incorporate temporal information. The particularity of LSTM units is the presence of a so called forget gate, which gives the unit the ability to forget its previous cell state for some values of the input. This enables the cell to react to sudden changes in the behaviour of the time series without taking into account information relating to a previous regime. More precisely, the layer is governed by the following equations:

$$
\begin{aligned}
f_t &= \sigma\left(U_f x_t + W_f h_{t-1} + b_f\right) && \text{(forget gate)} \\
i_t &= \sigma\left(U_i x_t + W_i h_{t-1} + b_i\right) && \text{(input gate)} \\
o_t &= \sigma\left(U_o x_t + W_o h_{t-1} + b_o\right) && \text{(output gate)} \\
\\
\tilde{C}_t &= \tanh\left(U_C x_t + W_C h_{t-1} + b_C\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t && \text{(cell state)} \\
\\
h_t &= o_t \odot \tanh(C_t)
\end{aligned}
\tag{12}
$$

Figure 22: Structure of the LSTM model



Source: Hochreiter and Schmidhuber (1997) & Amundi Institute.

where $\odot$ is the Hadamard product of two vectors. Since f_t, i_t, o_t are all gate structures to control the propagation of information, we should use the sigmoid function as the activation function with a return value between 0 and 1. The hidden layers $\tilde{C}_t$ and h_t always use the tanh function as the activation function for the same reason as explained in Remark 1. The cell state C_t acts like a long-term memory which can be forgotten if x_t and h_{t-1} reach certain values. This would make the signal f_t be equal to zero and thus filter C_{t-1} out of the formula. i_t and o_t, called the input and output gates respectively, act in a similar fashion. The input gate weights the signal $\tilde{C}_t$, which is a perceptron layer on the input x_t and hidden state h_{t-1}, whereas the output gate weights the cell state C_t before it becomes the new hidden state h_t. h_t is generally not used to compare directly with the true value y_t, but requires a feedforward layer like RNNs, as shown in Figure 19.

- **A cell state**, that represents the long-term memory of the network and goes through all hidden layers. We only perform some simple linear operations on it, so the information is easy to keep constant.
- **A forget gate**, that determines how much of the cell state from the previous time step is retained in the current time step.
- **An input gate**, that decides how much of the input of the current time step is saved to the cell state.
- **An output gate**, that controls how much of the cell state of the current time step is put into the output value.

The advantages and disadvantages of LSTM

As LSTM network is an extension of RNNs, it retains all the advantages of RNNs. Due to the existence of a cell state in the LSTM unit, the network has the ability to handle long

time sequences. In addition, we design three gates in LSTM units to regulate the flow of information into and out of the cell so that the network can maintain a long memory while also responding to sudden changes in the behaviour of the time series.

Due to the existence of the forget gate, LSTM networks are designed to prevent the vanishing gradient problem, which means that we can always find such a parameter update at any time step. However, as in the case of RNNs, the computational cost of LSTMs is also very high because the training process cannot be parallelized.

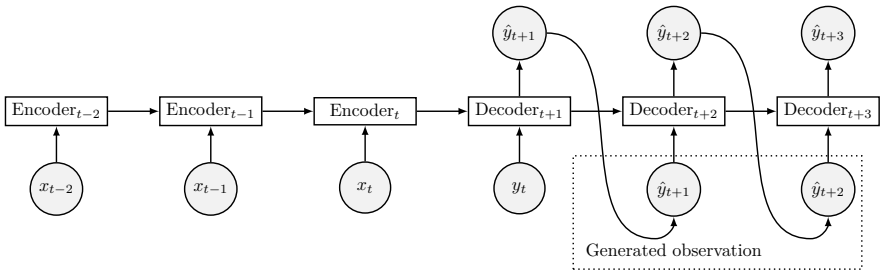
B.3 Free running and teacher forcing techniques

For a recursive encoder-decoder model shown in n Figure 4b, there are two basic training process: Free-running and Tearcher forcing.

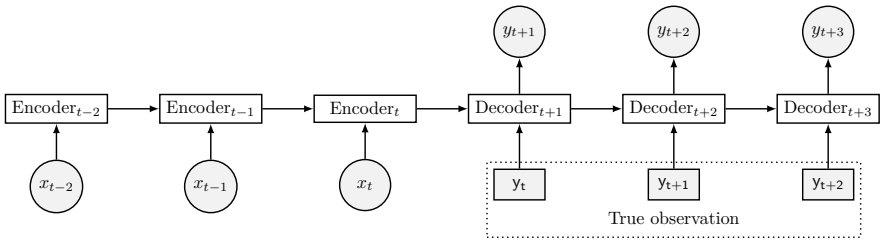
- Free running technique means that we feed generated observations into the model as input at each time step of decoder, as shown in Figure 23a.
- Teacher forcing technique means that we force feed ground truth labels into the model as input at each time step of decoder, as shown in Figure 23b.

Figure 23: Two ways of training recursive encoder decoder models

(a) Free running technique



(b) Teacher forcing technique



Source: Amundi Institute.

It can be seen that teacher forcing technique can prevent the model from being adversely affected by subsequent learning due to erroneous results from the previous time step, thus making the model more stable in the learning process and speeding up learning. However, teacher forcing technique has a drawback : the model is trained with only ground truth labels, and therefore, during testing, the model sometimes performs poorly if the sequence generated in the previous time step differs too much from the data used during training.

B.4 Batch normalization and Layer normalization

In deep learning, a learning rate that is too low can cause the model to take a lot of time to converge, and a learning rate that is too high can lead to the vanishing or exploding gradient problem. In practice, we may use normalization techniques to increase learning speed and reduce "Internal Covariate Shift"[13]. These normalization techniques, such as Batch normalization and Layer normalization, allow for much higher learning rates without the risk of divergence.

Batch normalization consists of normalizing the mini-batch input with its mean and standard deviation. We apply a batch normalization layer as follows for a minibatch $\mathcal{B}$:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$
$$\sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$
$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$
$$y_i = \gamma \hat{x}_i + \beta = \mathrm{BN}_{\gamma,\beta}(x_i)$$

where m is the batch size, γ and β are learnable parameters and ϵ is a small value.

It is worth mentioning that Batch normalization is performed on a small batch of samples (mini-batch) rather than all training samples and it is the normalization of each dimension of samples. Batch normalization is suitable for scenarios where each mini-batch is relatively large and the data distribution is close to each other, but it is not suitable for RNNs.

Layer normalization is the normalization of all dimensions of a single sample which is suitable for scenarios like small mini-batch and RNNs. We compute the layer normalization statistics over all the hidden units in the same layer as follows:

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij}$$
$$\sigma^2_i = \frac{1}{m}\sum_{j=1}^{m} (x_{ij} - \mu_i)^2$$
$$\hat{x}_{ij} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma^2_i + \epsilon}}$$

where m denotes the number of hidden units in a layer and ϵ is a small value.

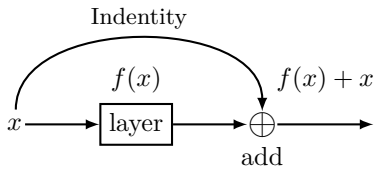
B.5 Residual Network

Residual Network (ResNet) was designed to address the degradation problem in deep learning, that is, the problem that increasing the number of layers in a neural network will sometimes lead to higher training errors. In theory, when we increase the depth of the network in a deep learning model, we can learn more complex structures from the dataset. However, we have found in practice that this is often not the case. As explained in He *et al.* (2016), if we already have a neural network that performs well in training and validation datasets, increasing its number of layers is equivalent to using these additional layers to

[13] This is the problem of Covariate Shift – the model is fed data with a very different distribution than what it was previously trained with – even though that new data still conforms to the same target function.

learn an identity mapping $f(x) = x$. The degraded accuracy of the deep learning model is due to the fact that these additional layers do not learn the identity mapping well. So, through a very clever design, the authors created a shortcut in the neural network to avoid this problem, which is the basic idea of ResNet.

Figure 24: The illustration of ResNet



Source: He *et al.* (2016) & Amundi Institute.

B.6 Mean-variance optimization approach

The single-period mean-variance model introduced by Markowitz in 1952 is one of the most widely used models in finance. This model used the concept: "*the investor does (or should) consider expected return a desirable thing and variance of return an undesirable thing.*" and can build an efficient portfolio that minimizes the risk for a given level of the expected return. We consider a universe of n assets and Markowitz (1952) formulated the investor's financial problem to build a fully invested portfolio as follows:

$$\begin{aligned} w^\star &= \arg\min_w \sigma(w) \\ \text{s.t. } &\begin{cases} \mu(w) \geq \mu^\star \\ \mathbf{1}_n^\top w = 1 \end{cases} \end{aligned}$$

where $\mu(w)$ and $\sigma^2(w)$ are respectively the expected return and the variance of the portfolio and $\mu^\star$ is the desired expected return. We can introduce a risk aversion parameter $\gamma \geq 0$ in order to rewrite the Markowitz problem in the form of a quadratic programming (QP) problem, which is easier to solve:

$$\begin{aligned} w^\star(\gamma) &= \arg\min_w \frac{1}{2} w^\top \Sigma w - \gamma w^\top \mu \\ \text{s.t. } &\mathbf{1}_n^\top w = 1 \end{aligned}$$

where μ and Σ are respectively the expected return vector and the covariance matrix of n assets. As described in Perrin and Roncalli (2020), in practice, we can introduce additional constraints in the mean-variance optimization framework to build a more realistic portfolio, such as no short selling, weight bounds, asset class limits, long/short exposure, etc. The framework can also be extended to other portfolio allocation problems, such as portfolio optimization with a benchmark, index sampling, turnover management, etc.

B.7 Risk budgeting approach

The fundamental principle of the risk budgeting (RB) approach is to allocate funds based on risk, rather than capital, as stated in Roncalli (2013). To achieve this, we introduce the concept of risk contribution, which is characterized as the contribution of each asset in the portfolio to the portfolio's overall risk. The portfolio manager defines a set of risk budgets

and then determines the weights of the portfolio such that the risk contributions are in line with the budgets.

From a mathematical point of view, a risk budgeting portfolio is defined as follows:

$$\begin{cases} \mathcal{RC}_i(x) = b_i \mathcal{R}(x) \\ b_i > 0, x_i \geq 0 \\ \sum_{i=1}^n b_i = 1, \sum_{i=1}^n x_i = 1 \end{cases} \quad \text{for all } i \tag{13}$$

where x_i is the allocation of Asset i, $\mathcal{R}(x)$ is the risk of the portfolio, which is typically the volatility of the portfolio, $\mathcal{RC}_i(x)$ and b_i are respectively the risk contribution and the risk budget of Asset i.

A route to solving Problem (13) is to transform the non-linear system into an optimization problem:

$$\begin{aligned} x^\star = \arg\min & \sum_{i=1}^n \left(\mathcal{RC}_i(x) - b_i \mathcal{R}(x)\right)^2 \\ \text{s.t.} \quad & x_i \geq 0, b_i \geq 0 \quad \text{for all } i \\ & 1^\top x = 1 \\ & 1^\top b = 1 \end{aligned} \tag{14}$$

However, Problem (14) is not a convex problem (Feng and Palomar, 2015), and the optimization has some numerical issues, particularly in the high-dimensional case, that is, when the number of assets is large. Roncalli (2013) shows a different approach to solving Problem (13) with the help of the logarithmic barrier method and the solution is:

$$x_{\text{RB}} = \frac{y^\star}{1^\top y^\star}$$

where $y^\star(c)$ is the solution of the alternative optimization problem:

$$\begin{aligned} y^\star(c) = \arg\min & \mathcal{R}(y) \\ \text{s.t.} \quad & \sum_{i=1}^n b_i \ln y_i \geq c \\ & y_i \geq 0 \quad \text{for all } i \end{aligned}$$

where c is an arbitrary scalar. This problem can be solved by the Newton algorithm (Chaves *et al.*, 2012) and the Cyclical Coordinate Descent (CCD) algorithm (Griveau-Billion *et al.*, 2013). In the special case of the equal risk contribution (ERC) portfolio, i.e. the risk budgets are the same ($b_i = b_j$, for all i, j), we have:

$$\begin{aligned} y^\star(\lambda) = \arg\min & \mathcal{R}(y) \\ \text{s.t.} \quad & \sum_{i=1}^n \ln y_i \geq c \\ & y_i \geq 0 \quad \text{for all } i \end{aligned}$$

Finally,

$$x_{\text{ERC}} = \frac{y^\star}{1^\top y^\star}$$

This portfolio allocation strategy is also known as the risk parity approach, which is the main alternative method to the traditional mean-variance portfolio optimization.