UNIVERSITA' COMMERCIALE LUIGI BOCCONI - MILANO

Department of Computing Sciences

Bachelor's degree in Mathematical and Computing sciences

for Artificial Intelligence

Deep Learning Methods for Asset Prices Estimation

Supervisor: Chiar.mo Prof. Claudio TEBALDI

Bachelor's degree Thesis of:
Matteo NULLI
SID. 3141748

Academic Year 2022 / 2023

Contents

1 Introduction **3**

2 What is Asset Pricing? **3**

3 Literature Review **6**

4 Innovations in the Field **7**

5 Methodology **7**

5.1 Generative Adversarial Method of Moments 8

5.2 Estimation 9

5.3 Deep Learning Model Architecture 9

5.3.1 Fully Connected Feed Forward Neural Network 10

5.3.2 Recurrent Neural Networks 12

5.3.3 LSTM - Long Short Term Memory Cells 14

5.4 Generative Adversarial Network Architecture 15

5.4.1 Digression on Convolutional Neural Networks 17

5.4.2 GAN architecture in the method 20

5.5 Model Training and Ensemble Learning 21

5.5.1 Dropout 21

5.5.2 Ensemble Learning 22

5.5.3 Training 22

5.5.4 Model Comparison 22

6 Empirical Results **23**

6.1 Data 23

6.1.1 Data Transformation 23

6.2 Model Performance 24

6.2.1 Detailed GAN Inner Workings 24

6.2.2 Cross Section of Individual Stock Returns 25

6.2.3 Hidden State Conditioning Impact 25

6.2.4 Variable Importance 26

7 Points of Reflection **28**

8 Conclusions 28

1 Introduction

Can Artificial Intelligence help humans predict future stock prices?
The objective of this research thesis lies precisely in answering this and many other such questions. This will be done by understanding and analysing how deep learning frameworks can be useful and effective when it comes to Asset Pricing Forecasting. While reviewing the most recent methodologies in the field of Deep Learning applications in Asset Pricing, this thesis will be taking inspiration primarily from the work of [Chen et al., 2023]. We will dive deeply into understanding the inner workings of Deep Learning Models as well as giving insights on which are the most crucial financial variables needed to accurately determine the future price of stocks. We will tackle questions like "how can Deep Neural Network architectures be able to capture the dynamic causalities that come into play when there is an asset pricing problem?, and, "Could some of these methods, which are already used in practice, ever become more powerful and accurate as computational power keeps rising?".
As the world plunges into the era of Large Language Models like ChatGPT, this paper tries to understand how portions of such big sensational technologies can be used in a very different manner and in a totally unrelated field such as the one of Finance.

2 What is Asset Pricing?

First and foremost, what exactly is asset pricing?
Essentially asset pricing is a technique which through functions of many variables, including economic and financial ones, determines the future prices of assets.
At its core is understanding patterns of asset's expected returns. Now crucial to asset pricing estimation is the stochastic discount factor. This is a random variable which has to respect the following equation:

$$E(\tilde{m}\tilde{x}_i) = p_i$$

where

- p_i is the initial price of the asset
- $\tilde{x}$ its the future cash-flow, i.e. the future price of the asset.

In other words the expected value of the product between the future price of an asset and this random variable (SDF) is equal to the initial price of the asset.
From this initial definition we can start to understand why SDFs are crucial when trying to

forecast the future price of a resource.
However as it is quite evident that SDFs change with respect to different assets.
The ideal goal for research in the field, for almost half of the past century, has been about finding a unique SDF, with which ALL assets could theoretically be valued.
As this thesis approaches to tackle this issue, it is better to list the four distinct problems in this quest:

1. SDF could be a function of a multitude of variables.
2. SDF's functional form may be too complex as well as not known.
3. SDF's dynamical structure might represent a liability in terms of risk exposure, and it may most likely be time varying on different economic conditions.
4. Low quality of signal in the risk premium of individual stock returns.

Intrinsic to the notion of SDF is the one of no arbitrage price. As per the 'Law of one price' the existence of a strictly positive SDF implies the absence of arbitrage opportunities in the market. The meaning of Arbitrage in the financial world is as better defined by [Harvard, 2023] the following:
"Arbitrage is an investment strategy in which an investor simultaneously buys and sells an asset in different markets to take advantage of a price difference and generate a profit.". After absorbing these basic notions it is time to dive deeper in the theories which regulate the world of asset pricing. Capital Asset Pricing Model (CAPM) and the No Arbitrage Pricing Theory (APT).
As simple as it may sound the CAPM is a model which helps investors evaluate the expected return of an asset. The main formula to evaluate the expected return of an investment in the model is $E(R_i) = R_f + \beta(E(R_m) - R_f)$ where R_f is the risk free rate, β is a weight on the $E(R_m) - R_f$ market risk premium. Indeed usually the value of β represents whether a stock is more or less risky then market. However, on top of other issues in the CAPM model, β is actually one of the most pressing, especially in nowadays financial world. Indeed in many instances the value of beta seems to have, over a prolonged time period, difficulties in explaining the behaviour of stocks. On top of this one of CAPM main assumptions is for the risk free rate to remain constant over the period of time, which in most cases does not hold true. On a side note, similar models which derive from the CAPM are Fama-French 3-factor model, which simply adds on top of CAPM the company size risk and the value risk factors, and Fama-French 5-factor model, which is an extension of the 3-factor one, adding the idea that companies having reported higher future earnings will record higher future stock price

returns and the fifth factor being investment.

A very valid alternative to CAPM is the Arbitrage Pricing Theory. Differently to its predecessor the APT clearly reveals why expected returns differ from each other. This disparity arises from the different level of exposure to the stochastic discount factor, also called pricing kernel, that each asset has. To better illustrate this, it might be best to break down the theory of Arbitrage Pricing. Envisioned in 1976, by Stephen Ross, the theory regards a sophisticated multi-factor technical model built upon on the relationship between financial asset's expected returns and their risk.

Before understanding the mathematics behind the theory, the no-arbitrage condition is the basic concept of this theory, but maybe even more basic than this is the notion of arbitrage opportunity, and both are crucial in understanding the following. As it gives name to the theory itself, the no-arbitrage condition is the capability of investors to build a portfolio through which, with the correct variable sensitivities to certain miss-priced securities, take advantage of an arbitrage opportunity. One of the theory's main assumptions resides in the belief that the asset's return is influenced by a multitude of factors. This is one of the main differences between this model and the Capital Asset Pricing Model (CAPM), which notably relies on the impact of a single element. Indeed, APT uses a multitude of systematic macroeconomic contributors when predicting the expected return. In particular it captures the sensitivity of each asset returns to the change in each variable. Other then the ability for investors to eliminate specific risk through well diversified portfolios, another fundamental underlying assumption, which names the theory, is the existence of no-arbitrage opportunities among well diversified portfolios and the important specification confirming that if arbitrage possibilities do exist then will be capitalized away by investors. A closer look at the formula to evaluate the expected return of an asset reveals the main difference with CAPM

$$E(R_i) = E(R_z) + E(I) - E(R_z) \times \beta_n$$

where $E(R_z)$ is the risk free rate of return, E_I the risk premia associated to the asset and β_n is the sensitivity of the asset's price to macroeconomic factor n, and these beta coefficients are usually estimated through a linear regression.

Other then those mentioned above the ATP paved the way for the use of statistical oriented models to predict stock returns and as such also the use of Machine and Deep Learning Techniques. On top of these another fundamental concept, and one which will be used recurrently in this thesis is the Sharpe ratio. It is a ratio comparing the return of an investment along with its risk. And it is given by the formula $S = \frac{E[R_p - R_f]}{\sigma_p}$, where R_p and R_f are respectively the return of the portfolio and the risk free rate, and σ is the standard deviation of the

portfolio's return. It is one of the most widely used methods in understanding the adjusted risk of an investment. A proper definition of what a good Sharpe ratio is does not exactly exist, however a Sharpe ratio > 1, is usually considered a "good" one, even though in most cases the higher the ratio the better.

3 Literature Review

As already specified in the introduction, probably the most cutting-edge and recent research in the field is the one done by [Chen et al., 2023] which tackles the issue of implementing reliable and robust Deep Learning models to asset Pricing. Similarly the research from [Giglio et al., 2021] stands out as not only being one of the most accurate and updated research papers of the field. Here they survey methodological contributions in the field of asset pricing and explore the world of Factor Models models by applying them to already in use reliable and robust asset pricing techniques. In a nutshell Factor models consists of statistical and machine learning techniques which base their share prices prediction on factors which can be subdivided in three macro areas: macroeconomic, microeconomic and implicit factors. However nowadays factors which are used in practise include: market, size, value, momentum and volatility. Each have their own reason of being present. The central idea of the paper is to understand how to link the always increasing ability of data-driven Machine Learning algorithms to extract thorough and methodical results which economic theories by them selves could not. As a consequence the fundamental purpose of Giglio, Kelly and Xiu can be summarized as follows:

1. Review contemporary methodological contribution in asset pricing research.
2. Examining how different aspects of asymptotic theory assist financial economists to find the most suitable models for their needs.
3. Compare techniques and analyze their strengths and weaknesses.

Maybe even more pioneer then the previously mentioned one is the research from [Gu et al., 2020], in which a thorough analysis of the behaviours of machine learning method in predicting US stock returns. Indeed most of their work serves as inspiration both to this thesis as well as to the work of [Chen et al., 2023].

4 Innovations in the Field

As we already partially described, it is particularly through [Chen et al., 2023] and [Gu et al., 2020] where we can see the potential of machine and deep learning in asset pricing.

The research from [Chen et al., 2023] in particular turns out to be the most innovative. On top of [Gu et al., 2020] they added the no-arbitrage condition as fundamental part of the deep learning algorithm. This implementation follows three fundamental steps:

- using a Feed Forward Network to explain the general functional form of the SDF
- capturing the time-variation of SDF by introducing a Long-Short-Term Memory network, which given a set of macroeconomic conditions, is able to understand their intricate dependencies.
- building test assets by identifying portfolios with most unexplained pricing information through a generative adversarial approach.

On top of this, their model, other then providing a brighter view of the pricing kernel and the sources of systematic risks, also outperforms [Gu et al., 2020] groundbreaking results. They were able to achieve this mostly thanks to three main systematic contributions, the primary being introducing a non-parametric adversarial approach to the estimation of financial data. As estimating the SDF account to solving a generalized method of moments problem, this novel adversarial approach opens the door to identifying the most meaningful conditions from an infinite set of possible moments. Additionally they were the first to introduce LSTM models as part of the network. This neural network structure, which will be later explained, allows to capture long term dependencies between time series input, allowing the model to recognize among others, business cycles. Last but not least their innovative formulation of the problem. In which the no arbitrage condition determines the components of the pricing kernel which hold a weak variance in the signal whilst having a high risk premia value.
In the next sections, this thesis will cover in depth the inner workings of the machine and deep learning methods used in [Chen et al., 2023] analysis.

5 Methodology

The model created in [Chen et al., 2023], can seem at a naked eye, though to understand, as such it might be best to break it down in a series of different components

- Generative Adversarial Method of Moments and Estimation

- Deep Learning Model Architecture
  - Feed Forward Networks
  - Recurrent Neural Networks
  - Generative Adversarial Network

In each of the following chapters, each method will be explained as a whole and in a second moment, contextualized in the use they make of it in their research.

5.1 Generative Adversarial Method of Moments

As previously mentioned, building the perfect model in order to find the weights of the SDF accounts to solving a method of moment problems.
In diving into the method it might be better to recall that when applying the Method of Moments the conditional no arbitrage moment condition needs to be satisfied

$$E[M_{t+1}R^e_{t+1,i}g(I_t, I_{t,i})] = 0$$

Where $g : R^p \times R^q \to R^D$ is a function, I_t and $I_{t,i}$ are its inputs, i.e. $I_t \times I_{t,i} \in R^p \times R^q$, and the former represents all the p macroeconomic variables, while the latter the q-firm specific characteristics, both are values which are found in the information set at time t.
M_{t+1} represents the Stochastic Discount Factor for t+1. and D is the number of moment conditions.
This unconditional moment condition formulation comes from the idea of interpreting it as pricing errors for a choice of portfolios and times which are determined by the functional g. When considering the largest mispricings we have to take into account the SDF portfolio weights $w_{t,i} = w(I_t, I_{t,i})$ as well as the risk loadings $\beta_{t,i} = \beta(I_t, I_{t,i})$. The objective of the problem is is finding those portfolios and/or times which are miss-priced the most and thus correcting the model in order to price those assets as well. This is done by optimizing the following min-max function

$$min_w min_g \frac{1}{N} \sum_{j=1}^{N} \left|\left| E\left[\left(1 - \sum_{i=1}^{N} w(I_t, I_{t,i}) R^e_{t+1,i}\right) R^e_{t+1,j} g(I_t, I_{t,j})\right]\right|\right|^2$$

This objective finds also a solid economic base in the research [Hansen and Jagannathan, 1997]. This paper shows that minimising the biggest possible error in pricing can be consistent with estimating the SDF, through least square distance, which comes the nearest to the true SDF. This problem however is particularly complex as there are an infinite number of candidate

moments, and at the same time there is no knowledge of which moments are actually able to identify the parameters. In addition, their parameter space has an infinite dimension, adding computational complexity.

5.2 Estimation

Given these consideration [Chen et al., 2023] choose to use a similar but slightly different loss function, as it gives a more stable SDF result, in the form of:

$$L(w|\hat{g}, I_t, I_{t,i}) = \frac{1}{N}\sum_{i=1}^{N}\frac{T_i}{T}\left|\left|\frac{1}{T_i}\sum_{t\in T_i} M_{t+1}R^e_{t+1,i}\hat{g}(I_t, I_{t,i})\right|\right|^2$$

As it is evident from the equation, it can be interpreted as a weighted average of the average pricing errors.

Now as it will be clear after we can divide the model in 2 components. The SDF Network, where for a given conditioning function $\hat{g}$ the model estimates the M_{t+1} portfolio weights by minimizing the pricing error loss $\hat{w} = L(w|\hat{g}, I_t, I_{t,i})$. And the Conditioning network where the conditioning function $\hat{g}$ will be constructed.

5.3 Deep Learning Model Architecture

The model architecture as briefly aforementioned is composed by two different parts which will compete against each other in a adversary fashion:

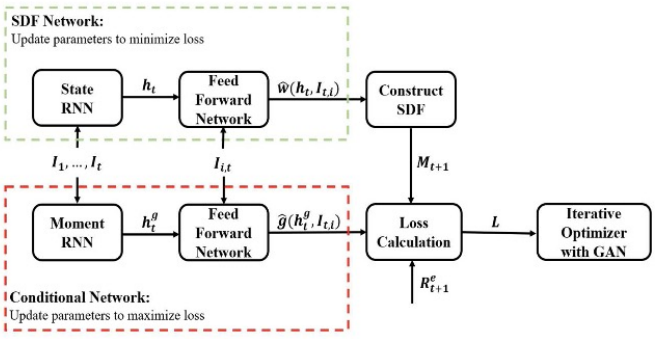

Figure 1: General Model Architecture

1. SDF Network

2. Conditional Network

Both components of the network have a similar process which can be roughly summarised as follows:

- Begin by giving as input to a Recurrent Neural Network the macroeconomic variables I_t
- Using Long-Short-Term-Memory layers extract the information and their dynamical structures.
- This information will serve as input to the next part of the network which consists of a Feed Forward Network.
- The FFN will perform some operations and its output will serve as the input of the next part of the Network which will vary depending on whether it came from the SDF Network or from the Conditional Network.

This is a lot of terminology and maybe it is time to better explain what some of them actually mean.

5.3.1 Fully Connected Feed Forward Neural Network

A Feed Forward Neural Network in a general setting is a function composition which in essence can be described by a directed a-cyclic graph. As a matter of fact, it is a non-parametric estimator for general functions, which, differently from other models of the same type, captures complex features between different types of data and most importantly is well suited to work with bigger sized datasets. This kind of model is one of the simplest in Deep Learning, mainly due to the absence of cycles and its structural linearity. The normal structure of FFN includes:

- Input Layer: each neuron of the layer represents one data entry.
- One or a series of hidden layers stacked together. They take the data and through a series of matrix multiplication processes and serve as a connector between input and output
- Activation Function: it is the part of the model which gives the non-linearity to the whole process. It is applied by every neuron to the average sum of its inputs. It would be best to say that if not for the non linearity of the activation function the neural network would essentially lessen to a linear model.

- Weights and Biases: these are the fundamental parameters of the network. The weights are present in every connection between the preceding neuron and the next. Biases on the other hand have one value per receiving neuron. Both of them are learned and updated while training the model to accurately depict the strength of the signal connections between one layer and the other.

- An output layer: final part of the model which produces the predictions.

In [Chen et al., 2023] specific case we can note both from the model architecture figure 1 and from the picture down below 2 how the input layer receives both macroeconomic inputs h_t as well as firm specific characteristics $I_{t,i}$.

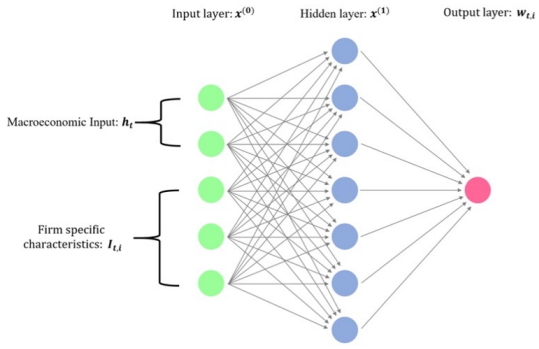

Figure 2: Feed Forward Neural Network inputs

The whole process starts combining the two inputs h_t and $I_{t,i}$ into one $x^{(0)}$. Then the process goes to the next step, i.e. to the hidden layer where the output is the result of a multiplication with a matrix of weights and the addition of a bias term. In mathematical terms the operation which will be performed to get the value of the first neuron of the hidden layer will be as follows:
The input values are $x^0 = (x_1^0, \ldots, x_n^0)$. The weight matrix $W^{(0)} = (w_1^0, \ldots, w_n^0)$ and the bias term b^0 are applied to the input value in the following way $(W^{(0)T}x^0 + b_1^0$ and then passed through an activation function $x^1 = ReLU(W^{(0)T}x^0 + b_1^0) = ReLU(\sum_{k=1}^{n} w_k^0 x_k^0 + b_1^0)$, yielding the hidden state values.
The activation function mentioned is the ReLU, Rectified Linear Unit which is $ReLU(x) = max(x, 0)$ and in a graph:

Coming back to the process, this x^1 is the vector of inputs to the hidden layer. This input will then be feed into another operation in order to obtain the output $y = w_{t,i} = W^{(1)T}x^1 + b_0^1$
Now in general neural network there may be multiple hidden units and the process would continue for each hidden unit until the end. Doing these calculations is helpful to understand

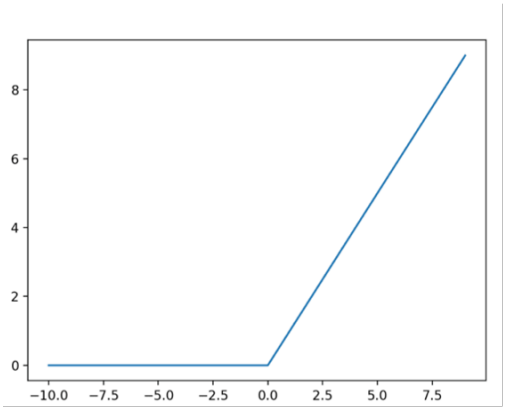

Figure 3: ReLU activation function

the deep power of neural networks and how in the blink of an eye they are able to perform those.

5.3.2 Recurrent Neural Networks

Recurrent Neural Networks are a type of Deep Neural Network used to process sequential data types. They are particularly useful, especially in this instance to estimate non-linear time-series dependencies.
Differently from a simpler FFN structure this architecture is designed in a way in which a storing process of intermediate inputs is in place. These stored values will then be compared with the successive outputs. This process is often referred to as a built-in feedback loop.
To break down the model we could summarize the initial structure as follows:

- x_t input
- h_t hidden state
- y_t output
- where $h_0 = \bar{0}$

Inside the cell the input x_t is firstly multiplied times its weight matrix $w_{h,x}x_t$ ($w_{h,x}$ is the weight of the connection from the input to the hidden state) and then concatenated with the previous iteration output $w_{h,h}h_{t-1}$ and a bias is added, as normally happens. This is then passed through a tanh activation function which results in the current state output $\rightarrow h_t = tanh(w_{h,x}x_t + w_{h,h}h_{t-1} + b)$.
Now at each time step potentially a loss can be evaluated, but before calculating the loss the output h_t needs to be multiplied by $w_{y,h}h_t$ where and $w_{y,h}$ is the weight of the connection

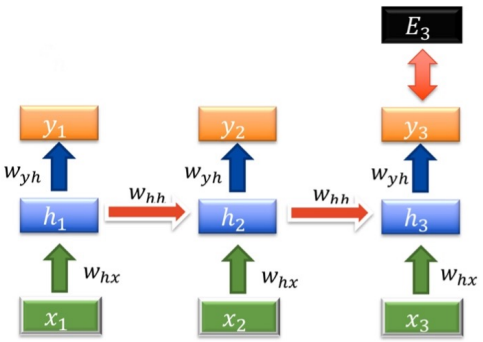

Figure 4: RNN model structure from [COMP 4329 University of Sydney, 2023]

from the hidden state to the output.

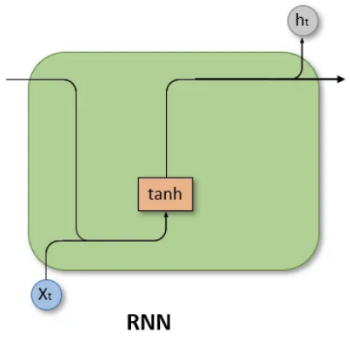

Figure 5: Inner RNN cell structure

However one of RNN's biggest fault happens during training, in particular when computing gradients.

Indeed we get

$$\frac{\delta h_{t+1}}{\delta w_{hh}} = tanh'\left(h_t + w_{hh}\frac{\delta h_t}{\delta w_{hh}}\right)$$

and by recalling that $h_t = tanh\left(w_{hh}h_t - 1 + w_{hx}x_t + b_h\right)$ when computing the next iterations of gradient, the problem becomes quite clear:

$$\frac{\delta h_t}{\delta w_{hh}} = tanh'\left(h_{t-1} + w_{hh}\frac{\delta h_{t-1}}{\delta w_{hh}}\right)$$

The value of this repeated multiplication of tanh functions heavily depends on the values of the weights, indeed as more iteration are taken into consideration, i.e.

$$\Rightarrow \frac{\delta h_{t+1}}{\delta w_{hh}} = tanh'\left(h_t + w_{hh}tanh'\left(h_{t-1} + tanh'\left(\ldots\right)\right)\right) \sim w_{hh}\left(tanh\right)' w_{hh}\left(tanh\right)' w_{hh}\left(tanh\right)'$$

this process will inevitably make the model incur in either Vanishing Gradients or in Exploding Gradients.
The former happens when $0 < w_{hh} < 1$ as

$$w_{hh} \, (tanh)' \, w_{hh} \, (tanh)' \, w_{hh} \, (tanh)' \, w_{hh} \, (tanh)' \ldots \rightarrow \; 0$$

Whilst the latter when w_{hh} is much larger as

$$w_{hh} \, (tanh)' \, w_{hh} \, (tanh)' \, w_{hh} \, (tanh)' \, w_{hh} \, (tanh)' \ldots \rightarrow \; \infty$$

5.3.3 LSTM - Long Short Term Memory Cells

To fix this problems researches developed in idea [Hochreiter and Schmidhuber, 1997] called Long Short-Term Memory also known as LSTM.
LSTM is essentially a specialized recurrent neural network. This kind of model can be seen as a hidden Markov model extension that employs nonlinear transition function, while at the same time being able to model long term temporal dependencies.

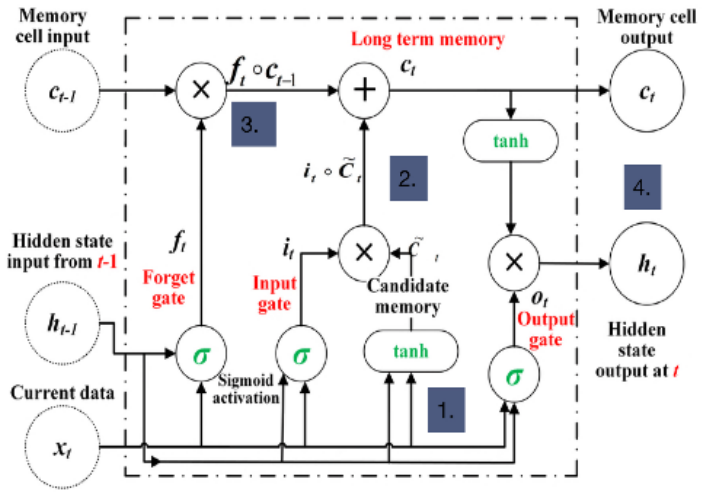

Figure 6: LSTM Model

The main blocks of LSTMs are the memory cells that act as our "memory" about the sequence of inputs we have processed thus far. They can selectively remember important information and forget irrelevant information. The three main components that allow it do so are:

- Forget Gate: determines what information to discard from the memory cell through a vector with values ranging from 0 to 1. Values close to zero will be "forgotten";

- Input Gate: determines what new information to store in the memory cell through a similar vector;
- Output Gate: controls the flow of information from the memory cell to the current hidden state. Similar functioning as the previous gates.

Firstly all gates and the input of the cell receive the same value which is a concatenation between the previous iteration output h_{t-1} and the new input word x_t, this concatenation will be called x. Each gate have normally a sigmoid activation function, which does the job of representing whether a gate is closed or open, the closer to 1 the more the input will be taken into consideration, the closer to zero the opposite. On the other hand, the normal input to the model has *tanh* as activation function. The concatenated input x after getting multiplied with a weight matrix W_x is then passed through a *tanh* and is then multiplied element wise with the result of the following operation from the input gate: $\sigma(W_{xi}\, x)$ sigmoid function, giving rise to lets call it $g = tanh(W_x\, x) \cdot \sigma(W_{xi}\, x)$. At the same time $\sigma(W_{xf} x) = f_t$ the forget gate f_t gets multiplied element wise with the previous memory c_{t-1}. Right after $f_t\, c_{t-1}$ this gets concatenated with the element wise multiplication from the input gate and the normal input, the one we called g which gives rise to c_t, the current time memory, which is both stored away to be used in next iteration and passed through a *tanh* activation. This output is then to be multiplied element wise with the output of the output gate, which is $\sigma(W_{xo}x)$, giving rise to $h_t = tanh(c^t) \cdot \sigma(W_{xo}x)$.
It is important to bear in mind that this is only one cell and that usually models are comprised of more LSTM layers staked together, in order to achieve maximum accuracy.
In this case the LSTM cells are used to cope with both a high-dimensional feature space as well as a more complex functional representation of states. At the same time keeping in memory the long-term dependencies in the data. On this matter, something which will be later covered will actually be how the LSTM can effectively extract business cycle patterns using deviations from long-term mean.

5.4 Generative Adversarial Network Architecture

This revolutionary model was firstly introduced by [Goodfellow et al., 2014]. The idea behind this architecture can be exemplified using the following "game". Counterfeiter-police game between two components:

a. Generator G

b. Discriminator D

Depending on the task the generator creates fake images/text, etc..
The Discriminator is trained using both real images and fake images generated by the generator, and his aim is to correctly classify real and generated images, while the Generator's objective is fooling the Discriminator into choosing as real image some image generated by the Generator.
This thought process can be explained by looking at the loss function of the model:

$$min_G max_D V(G, D) = E_{x \sim p_{data}(x)}[log(D(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

It is a min max problem. The first part being $E_{x \sim p_{data}(x)}[log(D(x))]$ the discriminator's predictions on the real data, while the second is the discriminator's prediction on fake data $E_{z \sim p_z(z)}[log(1 - D(G(z)))]$. Naturally following the Discriminator point of view he will try to maximize this sum as to obtain the best result.
More mathematically Discriminator's objective is to optimize the value of $D(x)$ as high as possible whether the input is real data $h = x$. This is because in the first part the higher the prediction on real data the better. It is imperative to notice that in the first part of the sum the generator is not involved. On the other hand the second part of the sum will be, for the discriminator about lowering the value of $D(G(z))$ as much as possible, hence resulting in a higher overall value from the expression $log(1-D(G(z)))$. This is where the Generator comes into play. Indeed since his goal is to fool the Discriminator he will try to do the opposite, i.e. maximize $D(G(z))$ hence minimizing the whole sum as a consequence.
The architecture can be visualized through picture 7.

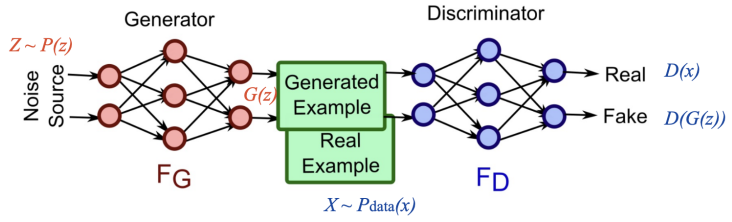

Figure 7: GAN Architecture from [COMP 4329 University of Sydney, 2023]

The most used Generator and Discriminator blocks in practise are Convolutional ones.
In order to understand what happens its important to explain what a Convolutional Layer is and how are they adapted to be used in a GAN architecture.

5.4.1 Digression on Convolutional Neural Networks

A general Convolutional Neural Network (CNN) architecture is made of a Convolutional Layer, a Pooling operation and a FC layer.

- **Convolutional Layer and Convolutional Operation**
  The whole process consists of 3 object mainly:
  - the input (say 6x6)
  - a filter (say 3x3)
  - a feature map (say 4x4)

  The filter (which we can see as a matrix of weights) goes through 3x3 subsets of the input and it is applied with an element wise multiplication.
  These multiplied elements are then added together and stored in the first element of the feature map.
  The process keeps going by moving the filter along the input until the feature map is full.

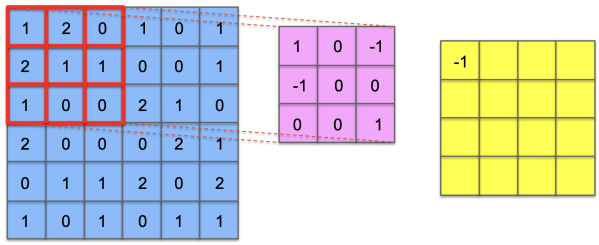

Figure 8: Convolution 1 from [COMP 4329 University of Sydney, 2023]

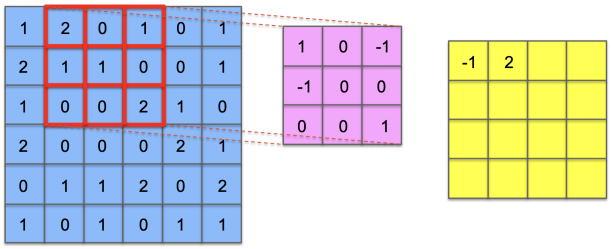

Figure 9: Convolution 2 from [COMP 4329 University of Sydney, 2023]

In figure 8 and 9 we can see that the way in which the filter moves along the input size is defined by a parameter called stride. In the above example the stride was $stride = (1, 1)$

meaning that it moved only by one space on the horizontal axis and by one space on the vertical when a row is completed.

There are many variations in the convolution operation: in theory and most often in practice multiple filters could be learned, this allows the model to learn different features by changing the weights of the filter.
Another common tool to use is padding, which essentially amounts to adding a layer of zeros around the input image. This enables the filter to capture in a better way the edges of the picture as the filter window goes through the edges more then it would have without it.
The example above was focusing only on 2D images as input, but most often in practise multiple channels are considered for one single image. Channels are multiple levels of brightness of different colours. In practise the channels which are often considered are RGB, that is the level of Red, Green and Blue in one picture.
In these cases what happens is that a different filter is applied to each channel. Or similarly as in 10 we get two filters for each input channels hence getting two feature maps, i.e. an output volume of 3x3x2.

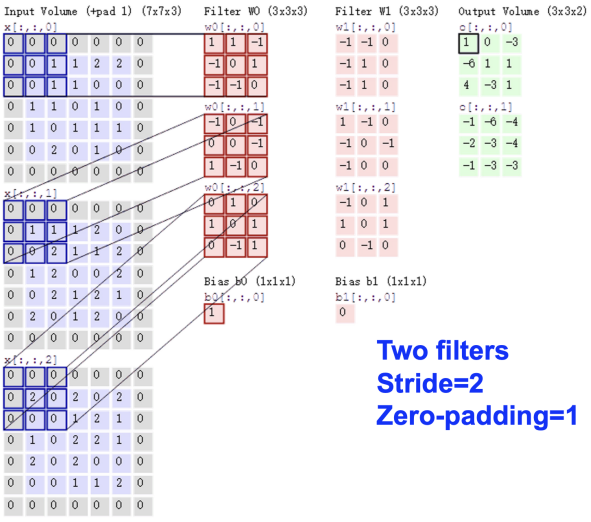

Figure 10: Convolution 3 from [COMP 4329 University of Sydney, 2023]

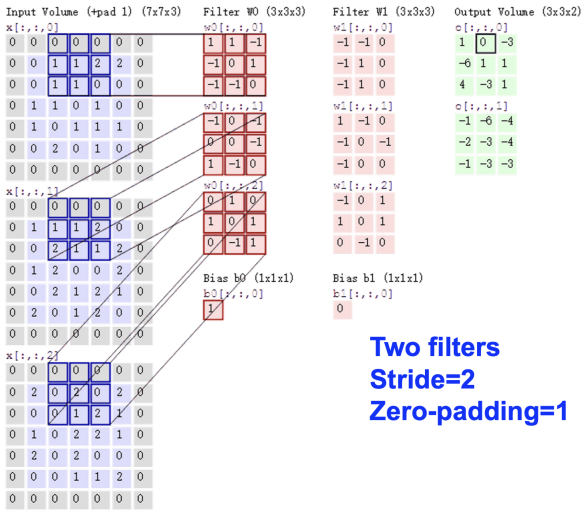

Figure 11: Convolution 4 from [COMP 4329 University of Sydney, 2023]

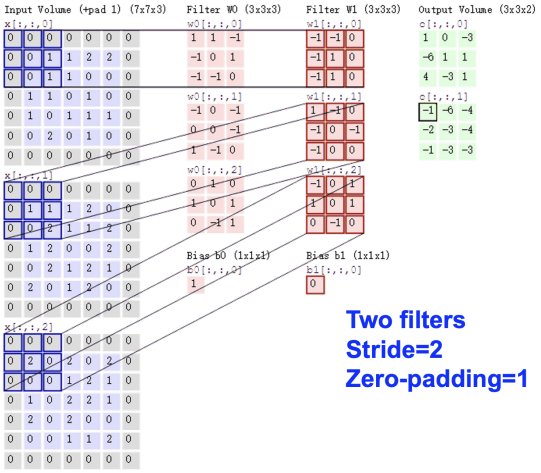

Figure 12: Convolution 5 from [COMP 4329 University of Sydney, 2023]

- **Pooling**

  Pooling is an operation used primarily to reduce the size of feature maps in a network. This is done for multiple reasons.

  1. For computational problems
  2. To prevent overfitting by keeping the most important bits of data
  3. Helping the representation becoming slightly invariant to small translations in the input
  4. It can be used to handle samples of different sizes.

This type of operation has a very similar behaviour to the convolution one. Meaning that it does have an input (which is the feature map resulting from the previous convolution) a filter and a resulting map which is called subsample map. By going through the input feature map and it slides the filter across, but the filter instead of having weights just performs an aggregate operation (max, average, etc...) and stores the resulting value of the operation. There are different kinds of pooling, however the most used in practise are max-pooling and average-pooling.

5.4.2 GAN architecture in the method

This adversarial approach in [Chen et al., 2023] is embodied in the two different parts of the model architecture 1. The Conditional Network's output, the function g and the SDF created by the above network i.e. M_{t+1} compete against each other, in a GAN fashion.
The conditional network acts as Discriminator trying to maximize the loss while the SDF network acts as the Generator by trying to minimize the loss. The steps in the training are the following three:

1. SDF minimizes the unconditional Loss
2. given M_{t+1} the parameters in the conditional network are optimized so to maximize the loss
3. Given the optimized conditional network the SDF Network is updated in order to minimize the conditional loss.

Differently to what was said before, here the structure of the Generator and Discriminator are an LSTM block, which has the job of understanding the hidden macroeconomic states and interdependencies along with a simple Feed Forward Network.
As a consequence of these processes the parameters are updated as follows

$$\hat{w}, \hat{h}_t, \hat{g}, \hat{h}_t^g = min_{w,h_t} max_{g,h_t^g} L(w|\hat{g}, h_t^g, h_t, I_t)$$

Where I_t are the macroeconomic variables
h_t and h_t^g are the summarized information from the macroeconomic inputs I_t outputted by the LSTM cells in the SDF network and Conditional Network respectively. Moreover its important to emphasise that h_t and h_t^g follow the criteria function of the two networks, indeed the first is the optimal hidden state which minimizes the price error, whereas the second does the opposite.

Lastly $\hat{w}$ and $\hat{g}$ are the outputs of the two networks, where the former is used to construct the M_{t+1} SDF while the latter goes directly into the loss calculation.

5.5 Model Training and Ensemble Learning

Training a Neural Network of this kind can sometimes turn out to be a much tougher task then expected. In doing so [Chen et al., 2023] used a very important regularization technique in order to avoid overfitting: that is dropout.

5.5.1 Dropout

Dropout is a regularization technique that allows large models to prevent overfitting.
When performing the forward pass, setting a dropout probability will activate only a random fraction of neurons in each layer.
The main difference with a standard feed forward operation is in the activation of previous layers, i.e. mathematically

$$z_i^{l+1} = w_i^{l+1} y^l + b_i^{l+1}$$

$$y_i^{l+1} = f(z_i^{l+1})$$

where is in dropout we have

$$r_i^l \sim Bernoulli(p)$$

$$\tilde{y}^l = r^l \cdot y^l$$

$$z_i^{l+1} = w_i^{l+1} \tilde{y}^l + b_i^{l+1}$$

$$y_i^{l+1} = f(z_i^{l+1})$$

This means setting to zero all the outputs of the "dropped" neurons. The objective is to force the network to not rely too heavily on a single unit when making the prediction. Furthermore, the weights will be scaled by a factor of p, again to avoid relying to heavily on specific units, which generally occurs during at testing.
A variant of this technique is **Inverted Dropout** in which at training a neuron is retained with probability p, and each neuron is scaled up by a factor $\frac{1}{p}$, whereas at testing the network is used as a whole but no scaling is applied. Differently from other standard regularization techniques such as l_1 or weight decay this one does not present the same disadvantages. Indeed Dropout is scale-free, i.e. it does not penalize large weight sizes, whereas Weight Decay does.

5.5.2 Ensemble Learning

The idea here is to train less complex model and then combining the simpler model to output a better final prediction. In Machine Learning the ensemble learning technique is usually performed by varying the initial parameters of the model, such as weights, biases.
The two main reasons for doing this are in the first place to lessen the effect of sub optimal fits and secondly to reduce the estimation of variance.
In [Chen et al., 2023] the average is done between nine different models, which accurately choose parameter from an optimal distribution. The final ensemble model outputs are calculated as a simple average over all 9 methods, each having same weight.

5.5.3 Training

The data used as input to the model is split into training, validation and test.
Validation Data is used for *hyperparameter tuning*. This consists in trying to optimize the values of parameters which are not optimized during training. In this category there are number of hidden units in each layer, number of hidden layers, value of p in dropout, and in this particular case the number of macroeconomic states as well as the structure of the conditioning network.
The optimal combination of hyperparameters are picked by maximizing the *Sharp ratio* of the M_{t+1} on Validation data. Moreover in their analysis the optimal hyperparameters for the Feed Forward Network are taken from the research of [Gu et al., 2020], this comes in handy when later comparing the results with their benchmarks.

5.5.4 Model Comparison

The evaluation of the performance of the model is done using the unconditional Sharpe Ratio of the SDF $SR = \frac{E(F)}{\sqrt{Var(F)}}$, the amount of explained variation in individual stock returns, i.e. $1 - \frac{\sum_{i=1}^{N} E[\epsilon_i]}{\sum_{i=1}^{N} E[R_i^e]}$ (where ϵ is a residual cross-sectional regression on the portfolio loadings) and the pricing errors, along with some other conventional characteristic sorted portfolios. The GAN model of [Chen et al., 2023] is compared with three other models, a linear factor model LS, a regularized linear factor model EN and a deep-learning forecasting approach FFN.
The four models have different results, in particular the linear factor models LS and EN can reach high Sharpe ratios but they cannot grasp the non linear dependencies in the SDF. On the other hand the FFN can capture this dependencies and achieve a high value for Sharpe ratio, but it most often cannot capture the loading portfolios structure correctly. That is where the GAN model with the help of the no-arbitrage condition makes the difference

and supports the model in dealing with low signal-to-noise ratio and producing an accurate depiction of the SDF loadings of stocks which have a lower risk premia value.

6 Empirical Results

Throughout this chapter it will be given a closer look to the how the previously defined model architecture by [Chen et al., 2023] will be used in practice. From which data will be used 6.1 and how it will be transformed to be fed as input to the GAN, to how the model compares to previous high standard asset pricing estimation methods 6.2 as well as comparing results between including or excluding various part of the GAN architecture 6.2.3.

6.1 Data

Data used in [Chen et al., 2023] is taken from monthly equity in the US from all securities on the center for Research and Security Prices (CRSP).
It goes over the period from January 1987 until December 2016. The splitting of training validation and test is done on 1967-1987 for training, 1987-1991 for validation and the rest for testing. The risk free rate which they make use of in order to calculate excess returns is the one-month treasury bill rates from Kenneth French Data Library.
Additionally the q-firm specific characteristics are actually 46 and are taken from Kenneth French Data Library. While the number of available stocks is around 30 .000, the ones which have all the firm characteristics are around 10.000.
Macroeconomic factors I_t are taken from different places. The first batch of 124 from FRED-MD database of which is extensively talked about in [McCracken and Ng, 2016]. The second 46 are taken by performing a cross-sectional median time series for each of the 46 previously mentioned firm specific characteristics. The final 8 are supplemented from a previous research [Welch and Goyal, 2008]. These last ones seem to have been decent predictors for the equity premium, but are not in the FRED-MD database.

6.1.1 Data Transformation

Data is subsequently transformed. First and foremost by ranking cross-sectionally each characteristic by month and converting them into quantiles. Another important tweak in the data is done by splitting each characteristic into two values. This is done by taking the previously rank-weighted average of stocks and separating the above median to the below median parts. A transformation is also applied to time series data regarding macroeco-

nomic factors. Transformation details are in the previously cites [Welch and Goyal, 2008] and [McCracken and Ng, 2016]

6.2 Model Performance

In this chapter it is displayed how the GAN model defined by [Chen et al., 2023] operates, its advantages and most of all its differences in behaviour when including or excluding some of the input values or parameters of the neural network.

6.2.1 Detailed GAN Inner Workings

Before explaining how the bigger model performs [Chen et al., 2023] try to so describe its inner workings by setting up a small illustrated example. Using only three different characteristics, that are size (LME), book-to-market ratio (BEME) and investment, leaving out all macroeconomic information. Some terminology to understand the example are UNC which denotes the model where the objective function has unconditional moment and test assets are individual stock returns. When the SDF M_{t+1} is allowed to depend on size and value we call the model UNC(SV), when adding investment the model is then called UNC(SVI). The GAN model constructed in their research, given his dual input structure has instead three different variations. GAN(SVI - SVI) where for both the SDF and the conditioning function g value, size and investment are considered, GAN(SVI-SV) where size value and investments are used for g but only size and value for SDF weights and lastly GAN(SV-SVI) which is exactly the opposite.

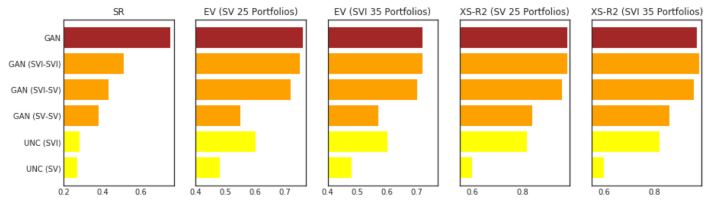

Figure 13: GAN illustration from [Chen et al., 2023]

On top of the previously described models, in 13 its clear how the full model where also macroeconomic factors and all the firm specific characteristics are taken into consideration can achieve a much stronger Sharpe ratio (SR), emphasising the importance of including more and more data.

The final take away from the example is how it is crucial to accurately choose the characteristics going into both the SDF weights as well as in the conditioning network, indeed the issue of determining an asset prices model for estimation cannot be parted from the accurate depiction of informative test assets.

6.2.2 Cross Section of Individual Stock Returns

Improvements of the GAN model are astonishing when compared to the simple FFN forecasting approach, the simple linear model LS and the regularized linear model EN, previously defined in 5.5.4. The model is able to catch over a 50% increase in the model performance with respect to the Sharpe ratio, as well as explain around 8% of the variance of individual stock returns, along with a 23% of the cross-sectional regression R^2 value. Down below in 14 we can clearly see the difference between GAN model from [Chen et al., 2023] and the previously described LS, EN, and FFN.

	SR			EV			XS-R^2		
Model	Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
LS	1.80	0.58	0.42	0.09	0.03	0.03	0.15	0.00	0.14
EN	1.37	1.15	0.50	0.12	0.05	0.04	0.17	0.02	0.19
FFN	0.45	0.42	0.44	0.11	0.04	0.04	0.14	-0.00	0.15
GAN	2.68	1.43	0.75	0.20	0.09	0.08	0.12	0.01	0.23

Figure 14: Model comparisons from [Chen et al., 2023]

where SR is the Sharpe ratio, EV is the expected variance and the $XS - R^2$ is the regression R^2 value.

6.2.3 Hidden State Conditioning Impact

The impact of conditioning on hidden macroeconomic states variables is also a factor to take into account. Indeed figure 15 shows how changing the way macroeconomic variables are fed into the model drastically changes the model performance.

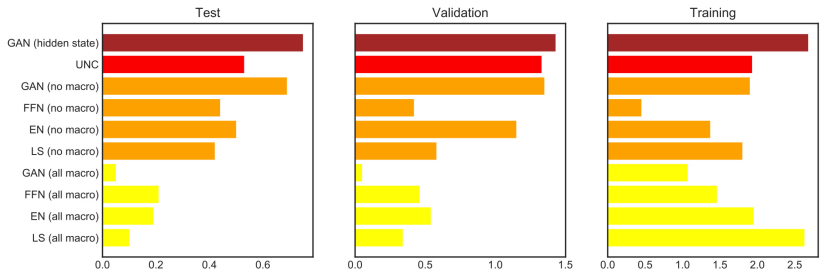

Figure 15: Importance Macroeconomic hidden states from [Chen et al., 2023]

GAN, FFN, EN, LS (no macro) are the models which only use firm specific characteristics, on the other hand GAN, FFN, EN, LS (all macro) are using all macroeconomic inputs but are not passing them through the LSTM structure, hence not accurately depicting the inner relationships between variables. Last but not least the GAN (hidden state), which uses the full model and indeed gives the best overall performance on the test set. From these set of results it is clear how the architecture of the LSTM turns out to be crucial in depicting the difference interdependencies between macroeconomic variables and turning them into actual information used in predicting both the SDF weights and the conditioning function values.

6.2.4 Variable Importance

In this section the Fama-French 5 factors from 2 will be used to compare results with the GAN factor. Their road map consists of first ranking by significance each firm-specific characteristic and macroeconomic variable, based on a sensitivity value which is evaluated using the absolute derivative of the gradient of the GAN model,

$$Sensitivity\,(x_k) = \frac{1}{C}\sum_{i=1}^{N}\sum_{t=1}^{T}\left|\frac{\delta w(I_t, I_{t,i})}{\delta x_k}\right|$$

where C is just a normalization constant, and a larger sensitivity denotes a larger effect of the variable x_k the SDF's weight.

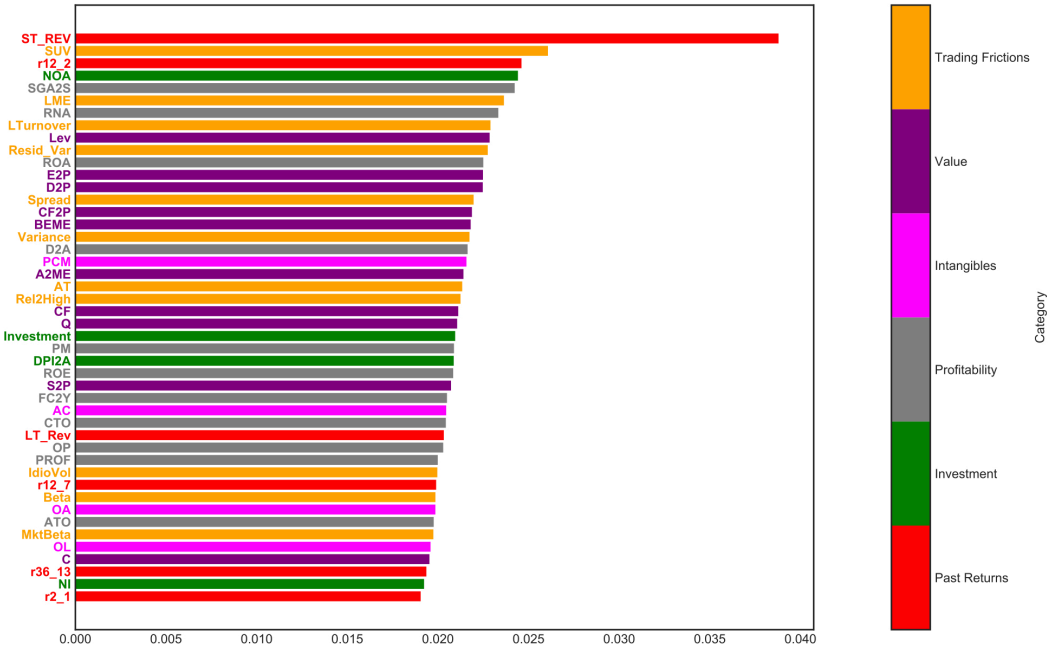

Figure 16: GAN variable importance [Chen et al., 2023]

The figure shows each of the 46 firm-specific characteristic's sensitivity normalized to sum up to one. Another quite clear insight from this table is the category at which each of the characteristic belong, notably for the GAN, which has 6 of these category, all categories are represented in the top 20 characteristics. And the three most relevant include the Short-Term Reversal (**ST_REV**) as well as Standard Unexplained Volume (**SUV**) and Momentum (**r12_2**).

Another variable importance related question is how much are hidden macroeconomic states tied with business cycles and/or collective economic activity?

Very much so, indeed the outputs of the LSTM models are closely linked with the aforementioned elements. In the following figure there are four unique combinations of macroeconomic variables. They are firstly given as an input to the model and then the corresponding output of the LSTM model (the hidden state) is plotted in the following:

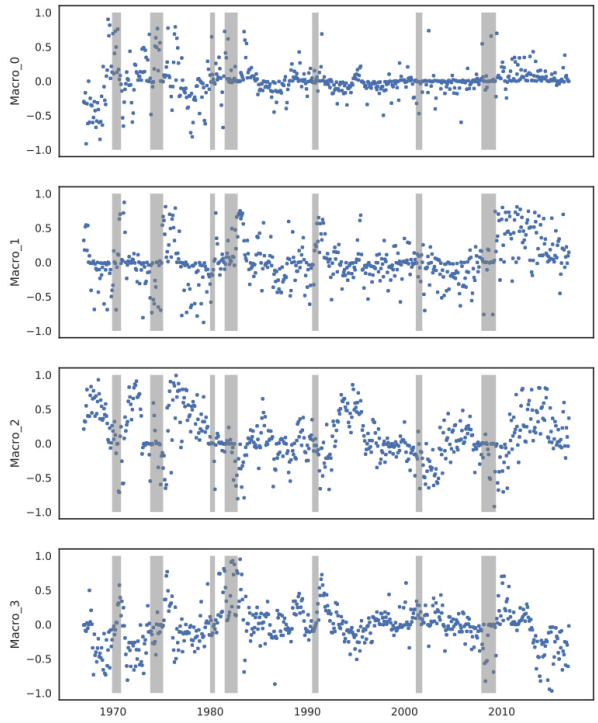

Figure 17: Plots of LSTM hidden state variables [Chen et al., 2023]

As a first observation we can clearly see especially in *micro_2* and *macro_3* the time series having spikes close to the grey areas, which are times of recession. On top of this the periodicity of the hidden state is also quite apparent from 17. Even though they seem to accurately follow a path it is simultaneously important to note how each macro factor does not have the same value at every single point in time, as it denotes a heterogeneous risk capturing.

7 Points of Reflection

Even though in the analysis only LSTMs and Generative Adversarial Network architectures are considered, in the future it might be sharp to conduct some analysis and experiments using Transformers architectures which are the basic to today's popular Large Language Models. An interesting idea for the analysis might be starting from newspaper articles, like Financial Times and other reliable sources of financial data, and using the transformers architecture to incorporate the sentiment of articles titles and/or corpus into the prediction of stocks.

8 Conclusions

It has now become quite clear how the methods used in this research thesis are able to accurately predict future stock prices.

This thesis shows that utilizing Deep Learning architectures not only improves performance of previous statistical based model, but also of economical models which were previously used to determine stock prices. On top of this the new and crucial no-arbitrage condition was shown to be effective in constructing efficient machine learning asset pricing models. Showing how a successful usage of Deep Learning techniques in Finance require a domain specific knowledge as well as the ability to develop complex models from a computer science stand point. Moreover it is crucial to take into account not only the macroeconomic condition of stock prices, but, most importantly, the complete period of the dynamic changes of the stock price and the final increments which might mislead the price forecasting. In addition to this, a surprising discovery is the seemingly linearity of stock estimation when eliminating outliers from consideration.

Furthermore, the model enabled us to understand the crucial factors that influence asset prices, detect discrepancies in stock valuation, and produce portfolios which are efficient in terms of conditional mean-variance. Besides, it internalizes an immense amount of time series data and especially through the LSTM cells and captures the non-linear dependencies and time-varying information into hidden meaningful states.

These considerations are only partially relevant but still there are some additional details which are more useful for both asset pricing researchers and investors and portfolio managers.

Regarding the former, the presence of new benchmark test assets turns out to be beneficial. This may be used to understand whether any future model will achieve better results. Additionally, the set of macroeconomic time series hidden state values could be used as an

input for other models. Pertaining to the latter, as the output of the model consists in the risk measure β and the weights of the SDF, any user could potentially assign a β and the corresponding portfolio weights to an asset even without having the whole time series data of that particular stock. This is because the model has already been pre-trained, and thus can be employed with any new stock.

In conclusion, it is just incredible the times we live in today. How much can a single model inference and predict using time series data on stock prices. It is indeed amazing how technologies, especially Artificial Intelligence, these days is been evolving and how much there is still left to uncover into its potentials.

References

[Chen et al., 2023] Chen, L., Pelger, M., and Zhu, J. (2023). Deep learning in asset pricing. *Management Science*.

[COMP 4329 University of Sydney, 2023] COMP 4329 University of Sydney, U. o. S. D. L. (2023). University of sydney unit of study, deep learning.

[Giglio et al., 2021] Giglio, S., Kelly, B. T., and Xiu, D. (2021). Factor models, machine learning, and asset pricing. *Machine Learning, and Asset Pricing (October 15, 2021)*.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., and Bengio, Y. (2014). Generative adversarial networks, 1–9. *arXiv preprint arXiv:1406.2661*.

[Gu et al., 2020] Gu, S., Kelly, B., and Xiu, D. (2020). Empirical asset pricing via machine learning. *The Review of Financial Studies*, 33(5):2223–2273.

[Hansen and Jagannathan, 1997] Hansen, L. P. and Jagannathan, R. (1997). Assessing specification errors in stochastic discount factor models. *The Journal of Finance*, 52(2):557–590.

[Harvard, 2023] Harvard, B. S. (2023). Harvard business school https://online.hbs.edu/blog/post/what-is-arbitrage.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[McCracken and Ng, 2016] McCracken, M. W. and Ng, S. (2016). Fred-md: A monthly database for macroeconomic research. *Journal of Business & Economic Statistics*, 34(4):574–589.

[Welch and Goyal, 2008] Welch, I. and Goyal, A. (2008). A comprehensive look at the empirical performance of equity premium prediction. *The Review of Financial Studies*, 21(4):1455–1508.