Distribution-Driven Portfolio Optimisation using Deep Learning

Canidate Number: QHSD5
University College London (UCL): Department of Statistical Science
Supervisor: Dr Purvasha Chakravarti
email: amjadsaidama@gmail.com

Date submitted - 11-09-2025

Contents

1 Introduction

1.1 What is a Portfolio and why Optimise it

A portfolio is a collection of related objects. For example, a collection of fruits: apple, banana, pear, orange, ... ext. Us humans typically prefer some fruits over others; these define our preferences, and we typically want to consume certain quantities of different fruits to maximise our benefits. The benefits derived from our preferences define our utility. As rational consumers, we assume our goal is to maximise utility. The solution at first seems trivial, just buy as much of what you like until your utility is maximised. Unfortunately, the solution to utility maximisation is not that simple, this is because in the real world we face constraints, such as how much money we have, our "wealth" and the law of diminishing marginal returns, meaning consuming an infinite amount does not satisfy optimlaity (maximum utility).

1.1.1 Apples and Oranges Anyone?

To better understand this problem, let's say we have two fruits, an apple and an orange selling for £0.50 and £0.80 respectively, we have a budget of up to £10. Our utility function that encodes our preferences is given by $2a + o^2 = U$, where a and o denote the quantities of apples and oranges. It is important to note that the utility function has an infinite number of level curves, in 2D these would be represented by a contour map or set of indifference curves at which a consumer is indifferent to consuming any given quantities of the products (apples and oranges) as they all yield the same U. We are effectively given a portfolio of two assets and told we face some constraints, formalising this problem we have

$$\max_{a,o} \left[2a + o^2\right] \quad \textbf{s.t.} \quad 0.50a + 0.80o \leq 10; (a, o) \geq 0 \tag{1}$$

Our utility function in 1 encodes our preferences; the function is linear in a and quadratic in o, meaning we only face diminishing marginal returns of consumption for oranges and not for apples. The solution to these kind of problems is calculated by finding where our budget constraint is tangential to the level curve of the utility function, which is solved using Lagrangian optimisation. The solution can then be compared to the edge cases (boundary conditions) to see if they are optimal.

Now, lets move to the financial setting where we have k independent and identically distributed (iid) random variables and some new utility function U which is a function of returns, risk, and quantities/weights w_i, $i = (1, \dots n)$, Our problem is to find the optimal set of weights that maximise our utility. It turns out this is a utility maximisation problem, particularly the mean-variance portfolio optimisation problem. The rest of this paper will discuss different formulations of this problem, their limitations, and propose novel redefinitions of the problem that overcome these limitations.

2 Literature Review

2.1 Portfolios and Mean-Variance Optimisation

For a fixed asset universe of size n, portfolio optimisation is the problem to decide assets, of which we have n - to hold (a portfolio) that have desired characteristics [1]. More formally, portfolio optimisation is the task of finding portfolio weights, $\mathbf{w} = (w_1, \dots w_n)$ corresponding to portfolio constituents (assets), that best satisfy some performance objective.

Quantitative models are usually employed to solve these complex problems. Markowitz's [2] mean-variance optimisation (MVO) portfolio is often considered as the inflection point

in quantitative-driven portfolio construction, forming the cornerstone of modern portfolio theory [3].

The MVO model assumes risk-averse agents facing a quadratic utility of form $U(x) = aw + bw^2$ with constants (a, b) and variable w, agents trade off mean for variance (risk), and want to be compensated with more return for taking on more risk. MVO assumes normally distributed returns, so for the n asset case, with corresponding return vectors $R_i \in \mathbb{R}^T$, $R_i \sim N(\mu_i, \sigma_i^2)$. Returns for any asset are calculated using simple returns, that is $r_{i,t} = \frac{p_{i,t} - p_{i,t-1}}{p_{i,t-1}} \in R_i$ and $t = (1, \dots, T)^{\forall}\ i = (1, \dots n)$. Generalising for n assets we have $\mathbf{R} = (R_1, \dots R_n) \in \mathbb{R}^{T \times n} \sim N(\boldsymbol{\mu}, \sum)$, Where $\boldsymbol{\mu} = \mathbb{E}[\mathbf{R}] = (\mu_1, \dots \mu_n)^\top$ and $\sum = \text{Cov}(\mathbf{R}) \in \mathbb{R}^{m \times m}$ where element $\sum_{i,j} = \text{Cov}(R_i, R_j)^{\forall}\ (i, j)$. The MVO problem can be defined in many equivalent ways. Let $\mathbf{R}_p = \sum_{i=1} w_i r_i$ be the portfolio return - portfolio return at any point in time is equal to sum of weighted returns across all assets -, therefore $\boldsymbol{\mu_p} = \mathbb{E}[\mathbf{R}_p] = \mathbf{w}^\top \cdot \boldsymbol{\mu}$ and $\text{Var}(\mathbf{R}_p) = \mathbf{w}^\top \sum \mathbf{w}$. We can now define the standard MVO portfolio optimisation problem ([2], [5])

Definition 1: Markowitz Mean-Variance Optimisation (MVO) problem.

$$\begin{aligned} \min_{\mathbf{w}} \left[\mathbf{w}^\top \Sigma \mathbf{w}\right] &= \min_{\mathbf{w}} \left[\sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j \sigma_{ij}\right] \\ \text{subject to} \quad \mathbf{w}^\top \boldsymbol{\mu} &= \mu_p, \\ \sum_{i=1}^{n} w_i &= 1 \end{aligned} \tag{2}$$

Where μ_p is the desired expected portfolio return and $0 \leq w_i \leq 1$, we can solve [1] to get the optimal portfolio weights for some desired return μ_p, $\hat{\mathbf{w}} \in \mathbb{R}^n$.

Reformulating the utility function by letting a and b equal the portfolio mean and variance we get an equivalent optimisation problem as in [2] for which an exact analytical solution exists.

$$\begin{aligned} U(\mathbf{w}) &= \mathbf{w}^\top \boldsymbol{\mu} - \frac{\lambda}{2} \mathbf{w}^\top \sum \mathbf{w} \\ \implies &\min_{\mathbf{w}} \left[-U(\mathbf{w})\right] \\ \text{subject to} \quad &\mathbf{w}^\top \mathbf{1} = 1 \end{aligned} \tag{3}$$

Instead of minimising risk for a fixed return, we now maximise return for a fixed risk.

2.2 Adaptations of the Mean-Variance Framework

2.2.1 Maximal Sharpe Ratio Portfolio

Advancements to the MVO model have been made, such the addition of cardinality constraints that limit the weighting of each asset or asset group [4]. MVO was further developed by Sharpe [5] who introduced the Capital Asset Pricing Model (CAPM). CAPM assumes investors choose portfolios along the capital market line (CML) [2], which is a linear function of risk free returns, market returns and portfolio volatility. The CML is represented in mean-variance space and reflects investors allocation between a risk-free asset and a market portfolio of risky assets.

Definition 2: CAPM and the CML, respectively.

$$\begin{aligned} \mathbb{E}[R_i] &= r_f + \beta_i \left(\mathbb{E}[R_M] - r_f\right) \\ \mathbb{E}[R_P] &= r_f + \frac{\mathbb{E}[R_M] - r_f}{\sigma_M} \cdot \sigma_P \end{aligned} \tag{4}$$

Where R_i is the asset return, $\beta_i = \frac{\text{Cov}(R_i, R_M)}{\text{Var}(R_M)}$ is the asset sensitivity relative to the market, R_P, R_M are the portfolio and benchmark returns respectively and σ_M and σ_P are their corresponding standard deviations. r_f is he risk-free rate.

The CML tells us that our expected portfolio return can be modelled as a linear combination of a risk-free asset plus our portfolio return variance scaled by the markets Sharpe Ratio.

The point at which the CML is tangential to the efficient frontier traced out by MVO problem is the maximal sharpe rate (often referred to as the risk premium) portfolio, which we yield by maximising the sharpe ratio w.r.t. weights subject to the constraint that weighted returns sum to unity.

Definition 3: The maximal Sharpe ratio portfolio in MVO.

$$\begin{aligned} &\min_w \left[-\frac{\mathbf{w}^\top \boldsymbol{\mu} - r_f}{\sqrt{\mathbf{w}^\top \sum \mathbf{w}}} \right] \\ &\textbf{s.t. } \mathbf{w}^\top \mathbf{1} = 1; \mathbf{w} \geq 0 \end{aligned} \tag{5}$$

This portfolio, is considered to be the "best" from a risk return perspective as it yields the greatest return per unit risk. (we will use this definition of MVO as a benchmark layer in our analysis). Solving [4] yields the the maximal sharpe ratio portfolio [Figure 1]

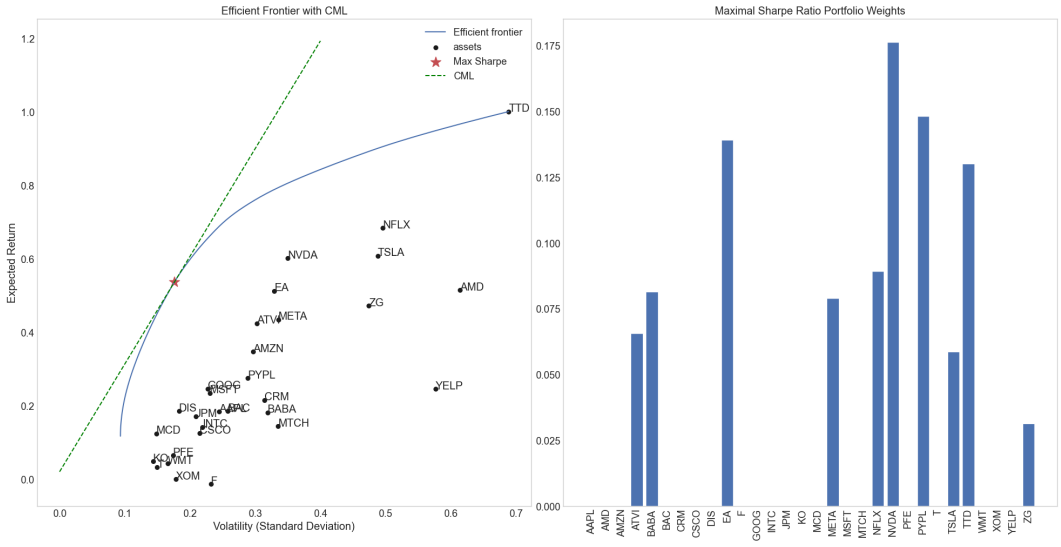

Figure 1: Result of maximal Sharpe ratio mean variance optimisation of our complete dataset, containing 30 assets and 1105 observations. **Left.** Efficient frontier, CML and maximum Sharpe ratio portfolio, **Right.** Asset weights from the maximum Sharpe rate portfolio.

Despite Constraint modifications, limitations of the MVO model exist when applied to real-world portfolio optimisation tasks. These include [3] ...

- **Estimation Error.** MVO relies on accurate estimators in order to find optimal weights.
- **Market non-stationarity.** Asset correlations tend towards unity during market crises, violating the stationarity assumption of MVO.
- **Non-Gaussian return distributions.** Real-world asset returns are not stationary and normally distributed, violating the multivariate normal distribution of returns MVO assumption.
- **Extreme parameter instability.** Small changes to parameter estimates (mean and covariance) cause large changes in final weights. Misspecification of expected returns dominates that of the variance by an order of magnitude.

2.2.2 Black-Litterman Allocation

Black and Litterman [6] present the Black and Litterman portfolio optimisation model (BL model) that addresses the most often cited weaknesses of MVO [7], parameter estimation error and parameter instability . The BL model takes a Bayesian approach to re-define the expected return vector, $\boldsymbol{\mu}$, given in the following form [8].

Definition 4: The Black-Litterman expected portfolio return vector.

$$\boldsymbol{\mu}_{BL} = \mathbb{E}[\mathbf{R}] = \left[\left(\tau \sum\right)^{-1} + P^{\top}\Omega^{-1}P\right]^{-1}\left[\left(\tau \sum\right)^{-1}\Pi + P^{\top}\Omega^{-1}Q\right] \tag{6}$$

Where $\mathbb{E}[\mathbf{R}]$, is the new-posterior expected return vector defined by the Black and Litterman approach. $\tau \in \mathbb{R}$ is the scaler tuning constant, $P \in \mathbb{R}^{K\times N}$ is the assets view matrix or picking matrix [8] with K views and N assets such that $K \subseteq N$, we can have **Relative Views**: "*Asset A will outperform asset B by 1%*", or **Absolute views**: "*Asset A will return 5%*". $\Omega \in \mathbb{R}^{K\times K}$ is the diagonal covariance matrix of error terms, ω, that represents the uncertainty in each view. Diagonal elements of the uncertainty matrix, Ω, are $\Omega_{kk} = \text{Var}(\omega_k) = p_k \sum p_k'$, this is the variance of the k'th views, $p_k \in \mathbb{R}^{1\times N}$, error term (The variance of the error term of the k'th view is equal to the variance of the k'th individual view portfolio of the asset view matrix P). $\Pi = \lambda \sum w_{mkt}$ is the implied equilibrium return vector which is a function of the risk premium, $\lambda = \frac{\mathbb{E}[\mathbf{R}]-r_f}{\sigma^2}$, the excess asset return covariance matrix, $\sum$ and the column vector of relative market capitalisation of the each asset $w_{mkt} \in \mathbb{R}^N$. $Q \in \mathbb{R}^K$ is the vector of views that tells us the expected returns for each view portfolio. To find $\hat{\mathbf{w}}$ we simply let $\boldsymbol{\mu} = \boldsymbol{\mu}_{BL}$ in 2 and solve using Lagrangian optimisation.

Despite the novel improvements, limitations of the BL approach exist, in fact BL inherits many of the same flaws of MVO and CAPM, such as market non-stationarity and non-Gaussian return distributions. A specific drawback of the BL model is ...

- **Model Bias.** View portfolios in the asset view matrix P are subjective so as are the expected returns of k'th view. This introduces bias if we have imperfect information about views and their returns.

2.3 Deep Learning for Portfolio Optimisation

In recent literature, deep learning models have gained popularity for their application in the portfolio optimisation problem [9]. These models rise in popularity can be attributed to

the fact that can be used to solve a wide array of prediction and classification tasks. These models only require a relationship between input and output to exist [9]. Authors in this field tend to employ one of two model methodologies: **Time-series variable forecasting** and **Continuous-time point estimate optimisation**. The former method typically concerns applying deep learning techniques to forecast either a scalar value, such as return and volatility, or predict the next market state (up or down). The model predictions are then used as inputs to a portfolio optimisation technique [9] outlined earlier. The later defines models that output portfolio weights directly, and the optimisation method is embedded into the training process, and allocations are updated continuously as new data arrives.

Continuous-time point estimations can be seen as superior models that output more optimal (unbiased, better performing) portfolio weights as optimisation objectives are embedded into the training process, requiring no separate statistical estimation step that suffers from model assumptions. Therefore, these models avoid explicit parameter estimation errors and can model non-linear dependencies, both of which are limitations of the MVO framework.

Zhang et al [10] propose a continuous-time point estimation model, specifically, the authors define an LSTM RNN agent optimised using the Sharpe ratio performance objective to find optimal portfolio weights [10]. Zhange et al use the expected return, standard deviation of return, Sharpe ratio, downside deviation of return, and sortino ratio as performance metrics of their model. These metrics are compared to those of other algorithms, such as reallocation strategies, classical MVO, maximum diversification, and stochastic portfolio theory of which the model outperforms, delivering the best performance [10].

Here, the price and return data for each asset are the only inputs into the single layer LSTM 64-unit network. LSTM's however are designed to deal with much richer feature data, so the authors may have underutilised the representational power of such a network. The obvious solution here would be to provide more features and modify the LSTM architecture to include more layers. However, this is at the expense of intuition, as our model becomes more "black-box". In fact much existing literature concerning deep learning for portfolio construction defines complex models, whose outputs are hard to explain and understand.

The Sharpe ratios use in part or entirely as a model objective function is a popular choice in existing literature ([11], [12]). Kim et al [12] employ a "**Decision by Supervised Learning**" (DSL) model, which is a form of continuous-time point estimate optimisation method as the model directly outputs portfolio weights. The authors use a cross-entropy loss that takes the model output weights and target weights from portfolio optimisation methods such as Max-Sharpe. By minimising the cross-entropy loss the DSL model approximates the optimisation process by learning a generalisable mapping from input data to weights. This method is proven to outperform classical optimisation methods such as standard MVO [12].

The main issue with this approach is that the optimization process relies on target weight estimates from classical portfolio optimisation methods such as standard MVO, therefore, this method is subject to all the same limitations and, therefore, can give a biased prediction for the optimal portfolio weights vector, $\hat{\mathbf{w}}$.

2.4 Fully Connected Networks (FCN's) and the Multi Layer Perceptron (MLP)

The multi layer perception (MLP) is a form of a fully connected neural network (FCN). These networks work by passing (feeding forward) a set of inputs through layers of pre-activation functions, $\mathbf{a}$, and post activation function's $\mathbf{Z}$, which are a function of parameters called weights, $\mathbf{W}$, and biases, $\mathbf{b}$ to produce a function estimate. The number of times the network applies sequential non-linear transformation to out data depends on the number of layers, L, in our network. These models are particularly useful as they can approximate any

function given enough hidden layers [13] (non input and output layers). Algorithm 1 details the feed-forward process

Algorithm 1 Forward Propagation Algorithm

Require:

- An L-layer neural network with non-linear activation function σ.
- Input data $\mathbf{x}$ (where $\mathbf{z}^{[l=0]} := \mathbf{x}$).
- Weight matrices $\mathbf{W}^{[l-1]}$ and bias vectors $\mathbf{b}^{[l]}$ for $l = 1, \ldots, L$.

1: **Initialise:** Set $\mathbf{z}^{[0]} = \mathbf{x}$.
2: **for** $l = 1, \ldots, L$ **do**
3: Compute pre-activation output vector (affine transformation) $\mathbf{a}^{[l]}$:

$$\mathbf{a}^{[l]} = (\mathbf{W}^{[l]})^\top \cdot \mathbf{z}^{[l-1]} + \mathbf{b}^{[l]}$$

4: Apply activation function σ to obtain $\mathbf{z}^{[l]}$:

$$\mathbf{z}^{[l]} = \sigma(\mathbf{a}^{[l]})$$

5: **Output:** The final output of the network is $\mathbf{z}^{[L]} = \sigma(\mathbf{a}^{[L]}) = f(x)$.

2.4.1 Gradient Decent

When first fitting the model, we initialise random parameter values, and as a consequence, our model estimates are suboptimal. To optimise our model we define a penalty function, $\mathcal{L}$, and a parameter update algorithm; this process is called backpropagation or backward-pass. The parameter update algorithm typically used is gradient descent, which works by differentiating a differentiable loss function with respect to some parameter evaluated at the current parameter guess and subtracting this from our current estimate to yield an updated parameter. The algorithm then loops this process for some layer, $l = (1, \ldots, L)$ until converging to the optimal parameters for that layer l. When applying backpropagation to all layers ,we say we have completed the backward-pass and out model is optimised.

In the context of our MLP, we have two parameters wish we wish to optimize, $\mathbf{W}^{[L]}$ and $\mathbf{b}^{[L]}$, To implement standard GD we can run the following algorithms...

$$\begin{aligned} \mathbf{W}^{[l]}_{n+1} &\leftarrow \mathbf{W}^{[l]}_{n} - \eta \nabla_{\mathbf{W}^{[l]}_n} \mathcal{L}(\mathbf{Y}, f(\mathbf{x}, \theta)) \\ \mathbf{b}^{[l]}_{n+1} &\leftarrow \mathbf{b}^{[l]}_{n} - \eta \nabla_{\mathbf{b}^{[l]}_n} \mathcal{L}(\mathbf{Y}, f(\mathbf{x}, \theta)) \end{aligned} \tag{7}$$

Here $\mathbf{Y}$ is the feature data (our set of known labels), and $f(\mathbf{x}, \theta)$ is the MLP model which is a function of input features $\mathbf{x}$ and the parameters $\theta = (\mathbf{W}, \mathbf{b})$. The gradient term in 7 is evaluated via applying the chain rule to the loss function output recursively until reaching the parameter in the l'th layer. The differentiation steps are generalised element wise as follows [14]

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w^{[l]}_{i,j}} &= \frac{\partial \mathcal{L}}{\partial z^{[l]}_j} \cdot \sigma'(a^{[l]}_j) \cdot z^{[l-1]}_j \\ \frac{\partial \mathcal{L}}{\partial z^{[l]}_j} &= \sum_{j=0}^{n_{l+1}-1} \frac{\partial \mathcal{L}}{\partial z^{[l+1]}_j} \cdot w^{[l+1]}_{ij} \cdot \sigma'(a^{[l+1]}_j), \text{ or if } l = L \implies \frac{\partial \mathcal{L}}{\partial z^{[l]}_j} = \nabla_{Z^{[L]}} \mathcal{L} \end{aligned} \tag{8}$$

The gradient of the loss with respect to the biases is identical to 7 excluding the $z_j^{[l-1]}$ which is the derivative of the the l'th layer pre-activation with respect to the l'th weight. Rather calculate the derivative of the l'th layer pre-activation with respect to the l'th layer bias term which evaluates to 1.

This process can be further generalised for all j elements; we then have the following identical formulation of the parameter loss gradients.

$$\delta^{[L]} = \nabla_{\mathbf{z}^{[L]}} \mathcal{L} \odot \sigma'(\mathbf{a}^{[L]}) \tag{9}$$

$$\delta^{[l]} = (\mathbf{W}^{[l+1]} \cdot \delta^{[l+1]}) \odot \sigma'(\mathbf{a}^{[l+1]}) \tag{10}$$

We have enough information to define the backpropagation algorithm or backward pass for a fully connected network (FCN).

Algorithm 2 Backward Pass Algorithm

Require:

- MLP parameters θ, input X, loss function $\mathcal{L}$, activation σ, learning rate η
- MLP forward pass

function BackwardPass($\theta, X, \mathcal{L}, \sigma, \eta$)
  Initialise: $\delta^{[L]} = \nabla_{\mathbf{z}^{[L]}} \mathcal{L} \odot \sigma'(\mathbf{a}^{[L]})$
  for $l = L, L-1, \ldots, 1$ **do**
    if $l = L$ **then**
      $\frac{d\mathcal{L}}{dz^{[l]}} = \delta^{[L]} = \nabla_{\mathbf{z}^{[L]}} \mathcal{L} \odot \sigma'(\mathbf{a}^{[L]})$
    else
      $\frac{d\mathcal{L}}{dz^{[l]}} = \delta^{[l]} = (\mathbf{W}^{[l+1]} \cdot \delta^{[l+1]}) \odot \sigma'(\mathbf{a}^{[l]})$
    Compute gradient of loss with respect to parameters of the l'th layer:

$$\frac{\partial \mathcal{L}}{\partial \theta^{[l]}} = \frac{d\mathcal{L}}{dz^{[l]}} \frac{dz^{[l]}}{da^{[l]}} \frac{da^{[l]}}{d\theta^{[l]}}$$

    for each number of updates $n = 1 \ldots N$ **do**
      Update parameters to learn parameter coefficients:

$$\theta_{n+1}^{[l]} \leftarrow \theta_n^{[l]} - \eta_n \cdot \frac{\partial \mathcal{L}}{\partial \theta^{[l]}}|_{\theta^{[l]}}$$

    Store final update:

$$\hat{\theta}^{[l]} = \theta_N^{[l]}$$

2.5 Long Short Term Memory (LSTM) Models

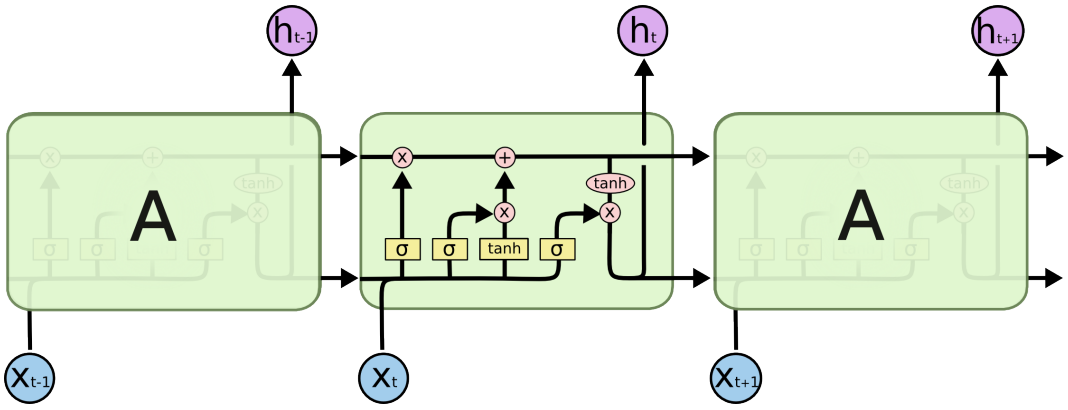

Figure 2: A single LSTM cell unrolled over 2 time periods [15].

Long-short-term-memory models, LSTM's, are a type of recurrent neural network (RNN) that overcomes the limitation of vanishing gradients [16]. The LSTM model does this by including a forget gate that regulates the information flow.

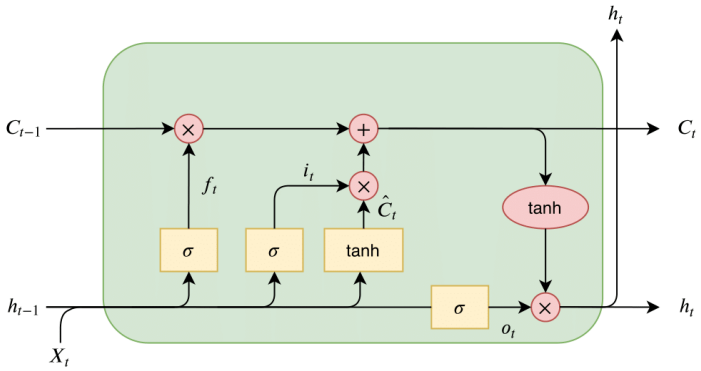

Figure 3: Single LSTM cell with inputs x_t: the input, h_t: cell output, C_t: cell state, f_t: the forget gate, o_t: the output gate, $\hat{C}$: the internal cell state [17].

An LSTM has three gates, **Forgate gate**, **Input gate** and **Ouput Gate**.

2.5.1 Forget Gate

The forget gate activation layer is

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Observations that enter the forget gate are not passed to the cell state. h here is the cell output or the hidden state, this is the short-term memory of the model. Each dimension of the hidden state represents a learned feature in the embedding space. b_f if the bias vector for the forget gate.

2.5.2 Input Gate

Here we decide what we are going to store in the cell state. The input gate has two components, i_t and $\hat{C}_t$. i_t is the input gate layer and we use it to decided which values we will update. $\hat{C}_t$ is a vector of new candidate values that we will add to the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\hat{C} = tanh(W_c \cdot [h_{t-1}, x_t] + b_C)$$

b_i is the bias weight and b_C is the bias vector corresponding to the estimated cell state.

2.5.3 Cell State

Here we decide on how to update the old cell state, C_{t-1} (The prior cell state, C_{t-1} represents the long term memory of the network). This is simply the sum of the forget gate (getting rid of information we wanted to forget) weighted by the prior cell state plus the new candidate values $\hat{C}_t$.

$$C_t = (f_t \cdot C_{t-1}) + \left(i_t \cdot \hat{C}_t\right)$$

2.5.4 Output Gate

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \cdot tanh(C_t)$$

The sigmoid layer, o_t, decides what part of the cell state we are going to output. We finally take the tanh() of the cell state C_t to bound the vaue between -1 and 1 and multiply the value by the sigmoid layer

3 Methodology

3.1 Research Aims

Portfolio optimisation describes the problem of finding weight quantities corresponding to each asset in a portfolio that maximise some predefined utility function. Existing formulations of the portfolio optimisation problem are limited by unrealistic real-world assumptions and overly simplified utility functions that fail to encode more complicated investor preferences. Our research problem is to propose a novel model that overcomes the limitations of existing literature by leveraging the ability od **continuous-time point estimation** Deep-Learning techniques, and statistical theory. We also require our models to provide superior performance over competing methods. We will do this by discussing suitable loss functions (Chapter 3.3), model architecture (Chapter 4.2) and portfolio constraints for our optimisation problem (Chapters 4.0.2 and 4.0.4).

3.2 Research Problem

Definition 5: Our proposed portfolio problem can my formalised as follows.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{R}\sim\mathcal{D}}[\, \mathcal{L}(\underbrace{f_\theta(\mathbf{R})^\top \mathbf{R}}_{\hat{R}}, R^*)\,] \quad \text{s.t.} \quad \sum_{i=1}^{n} |\hat{w}_i| = 1 \tag{11}$$

Where R^* represents the true label and $\hat{w}$ represents the weight estimates produced from $f_\theta(\mathbf{R})$. $\mathbb{E}_{\mathbf{R}\sim\mathcal{D}}$ is the empirical **risk**, which is an unbiased estimator of the **true risk**; the expectation is taken over $\mathbf{R} \sim \mathcal{D}$, meaning the distribution of $\mathbf{R}$ is unknown, hence requiring us to calculate the empirical risk. Without knowing the distribution, there is no analytical form of the density function. $\mathbf{R} \in \mathbb{R}^{T\times n}$ denotes the set of n, T dimensional return vectors, with each $i = (1, \dots n)$ defines asset i's simple market returns.

$\mathcal{F}_\Theta$ is the universal hypothesis class representing the set of all prediction models; this set includes linear models, non-linear models, ensemble models, neural networks, and more. We focus strictly on the set of neural network models, $\mathcal{F}_\theta \subset \mathcal{F}_\Theta$. Our specific model choice is $f_\theta \in \mathcal{F}_\theta$, the optimal empirical value this function can take is $f_{\hat{\theta}}$, which is attained via solving 11. The best optimal model within our set of models $\mathcal{F}_\theta$ is $f^*_{\theta^*}$, and the best theoretically attainable model is the Bayes predictor f^*, which lies on the boundary of the set $\mathcal{F}_\Theta$.

3.2.1 Error Decomposition

It is important to know the error of our model. It is unlikely that our optimisation process will find the global minimum and therefore attaining $\hat{\theta}$ is not possible. Rather, our optimisation process will yield parameter $\tilde{\theta}$, the observed optimal parameter. So the observed optimal model is $f_{\tilde{\theta}}$.

We will define $R()$ as the risk function. If this function takes estimates (denoted by $\hat{}$ or $\tilde{}$) then $R()$ is the empirical risk. The difference here is to communicate the use of population and sample data and information used in model construction (e.g. we would use empirical risk to measure the error of our models, making use of finite sample data). Using the above notation and definitions discussed so far in this sub-chapter, we can define the **overall error at scale**.

$$\begin{aligned}\mathbb{E}\left[\hat{R}(f_{\tilde{\theta}}) - R(f^*)\right] &= \mathbb{E}\left[R(f^*) - R(f^*_{\theta^*})\right] + \mathbb{E}\left[R(f_{\hat{\theta}}) - R(f^*_{\theta^*})\right] + \mathbb{E}\left[R(f_{\hat{\theta}}) - R(f_{\tilde{\theta}})\right] \\ &= \text{Approximation Error} + \text{Estimation Error} + \text{Optimisation Error}\end{aligned} \tag{12}$$

The **approximation error** is calculated as the expected difference in model risk between the Bayes predictor and the best optimal model in our function family. The **estimation error** is the difference in model risk between our optimal function (the function itself is not optimal, this is $f_{\tilde{\theta}}$) and the best optimal model in our function family. The last part of the overall error at scale is the **optimisation error**, which is the difference in risk between our optimal model and the observed optimal model discussed above.

We also assume that we choose the most suitable models within our function family, therefore minimising the estimation error. If we also assume that our parameter optimisation algorithms find a global minimum, then our optimisation error is zero. Given the above holds, if we out function family $\mathcal{F}_\theta$ contains the Bayes predictor, our approximation error is also zero, although this is very difficult to prove as the Bayes predictor is unknown.

3.2.2 Research Pipeline

All our discussed supervised based machine learning based models will follow the same general process detailed below ...

1. **Construct a Feature Vector.** We start by defining our set of independent variables (features) that we wish to use to predict labels. Our feature are the return vector $\mathbf{R}$ and our labels are the target returns R^*.

2. **Choose Model.** At this stage, we choose our specific model type, $f \in \mathcal{F}_\Theta$, that we will use for the supervised learning task. We focus on neural network models, $\mathcal{F}_\theta$. We will

define two such model, f_{θ_1} and f_{θ_2} (Chapters 4.2.1 and 4.2.2)

3. **Choose a Loss Function.** Define a function $\mathcal{L}(R_P, R^*)$ that takes model predictions (optimised model in-sample returns $\hat{R}$) and true labels (target returns R^*) as input. These functions are typically called loss functions and quantify how good of a job our model has done at predicting the true labels. The lower the loss, the better the model. (Chapters 3.3)

4. **Compute Empirical Risk.** We compute the error of our model, $R(f_{\hat{\theta}})$, using the average of the loss function over all predictions (Chapter 3.2.1).

5. **Train Model.** This is the "Training" or "Optimisation" process in the machine learning process. Our goal is to minimise our loss, which we achieve by minimising our empirical risk. This is exactly equation 9, and is called **empirical risk minimisation** (ERM).

6. **Predict.** After finding the best possible model, we will use the model as a prediction tool by fitting the model on unseen data. In our case, we would feed through the out-of-sample prior days' asset returns, which ensures our model does not have lookahead bias (making predictions based on data that was not actually available at the time) leading to biased performance (Chapters 5) .

3.3 Objective Functions

Instead of defining a model loss function that is a portfolio performance objective or measure of similarity between target weights, we investigate loss functions that measure return distribution similarity. This family of loss functions compares the return distribution of our model to some pre-defined target known with certainty. This way our model learns to map inputs to weights that correspond to some known return distribution of choice we wish to replicate. This may be important to an asset manager wishing to build a portfolio with expectation and variability of return in mind. Furthermore, this approach allows us to specify a target expected return and risk, not possible in 2, 3 and 6, which only have 1 free variable, either return or risk.
The family of loss functions we define that satisfy this property is are distance metrics, $d : V \times V \to \mathbb{R}$ (Where V is a vector space) that are able to measure distribution similarity. A valid metric, d, must be **non-negative**, **symmetric**, and satisfy the **triangle inequality** [18]. We explore three different distance metrics, the L_2^2 divergance, the Maximum Mean Discrepency (MMD), and the Wassertein distance.

3.3.1 L^p distance and L_2^2 Divergence

The generalisation of the L^p distance is ...

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_p &= \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \\ &= (\langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle)^{\frac{1}{p}} \end{aligned} \tag{13}$$

and as $\text{Lim}_{p\to\infty}[\|\mathbf{x} - \mathbf{y}\|_p]$, i.e. as p tends to ∞ the p-norm is, the maximal norm ...

$$\|\mathbf{x} - \mathbf{y}\|_\infty = \max_{1 \le i \le 1} |x_i - y_i| \tag{14}$$

Note that $p \neq d$, and high p makes d more sensitive to outliers, as the distance becomes dominated by the largest component difference. In our research, we will set $p = 2$, then 13 simplifies to the Euclidean distance. For the Euclidean distance to measure distribution similarity, we must make the metric a function of the PDF of the corresponding random variables and multiply by a constant, here $\Delta x = \frac{\max(x)-\min(x)}{n}$.

Definition 6: The L_2^2 empirical divergence.

$$d(P(\mathbf{x}) - Q(\mathbf{y}))^2 = \|P(\mathbf{x}) - Q(\mathbf{y})\|_2^2 = \sum_{i=1}^{n}(p(x_i) - q(y_i))^2 \tag{15}$$

Where

$$\text{Gaussian Kernel Density Estimator} = p(x) = q(x) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot \exp\left\{\frac{x^2}{2}\right\} \tag{16}$$

Equation 15 is a valid finite sample representation of the L_2^2 divergence. We know proceed with a sketch proof to prove this statement.

Radon-Nikodym derivatives (RND's) of continuous probability distributions P and Q are $\frac{dP}{d\mu(x)} = p(x)$ and $\frac{dQ}{d\mu(x)} = q(x)$, where $p(x)$ and $q(x)$ are densities of distributions P and Q. P and Q are also probability measures, defined over some domain $\mathcal{X}$ with $P(\mathcal{X}) = 1$ and $Q(\mathcal{X}) = 1$, which is a requirement for a valid probability distribution. $\mu(x)$ is a probability measure which is basically a set with associated properties. We can now define the continuous case of the L_2^2 divergence [19].

$$\int_a^b (p(x) - q(x))^2 \, d\mu(x) \tag{17}$$

Where $\mu(x)$ in 17 is the Lebesgue measure, dx. 17 is also a Riemann Integrable function. We know this because densities p and q are continuous on the interval $[a, b]$, so the distance function $f(x) = d(x, x)$ is continuous on $[a, b]$, hence Riemann Integrable. To obtain a finite sample approximation equal to that in 15 we let $\mu(x)$ equal the counting measure, which allows us to express 15 using Riemann sums.

$$\int_a^b (p(x) - q(x))^2 \, d\mu(x) \approx \sum_{i=1}^{n}(p(x_i) - q(y_i))^2 \cdot \Delta x$$

The limit of our function $d(x, x) : [a, b] \to \mathbb{R}$ as the partition of the domain goes to zero is the continuous L_2^2 divergence

$$\lim_{\max(\mathrm{P}|\mathrm{x}|)\to 0} \left[\sum_{i=1}^{n} f(x^*)\Delta x\right] = \int_a^b (p(x) - q(x))^2 \, d\mu(x)$$

3.3.2 Maximum Mean Discrepency (MMD)

Gretton et al [20] define the maximum mean discrepancy (MMD) as a test statistic equal to the largest difference in expectations over functions in the unit ball of a reproducing kernel Hilbert space (RKHS). The MMD is designed to test for similarity between two sets

of independent and identically distributed (iid) samples of random variables, X and Y, with associated probability density functions p and q, the MMD thus tests if $p = q$.

Let $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to \mathbb{R}$ where $\mathcal{X}$ is a topological space (a set equipped with properties, we will consider the standard Euclidean topology/geometry such that $\mathcal{X} \in \mathbb{R}^d$) with respective borel probability measures P and Q. The MMD is defined as ...

$$\text{MMD}[\mathcal{F}, P, Q] := \sup_{f \in \mathcal{F}} \left(\mathbb{E}_X[f(X)] - \mathbb{E}_Y[f(Y)] \right) \tag{18}$$

A biased estimate of the MMD can be obtained by replacing the population expectations with empirical expectations computed on the samples $\mathbf{x} = (x_1, \dots x_m)$ and $\mathbf{y} = (y_1, \dots y_n)$, here $\mathbf{x}$ and $\mathbf{y}$ are $\mathbb{R}^{m=n}$ sets of observations of the random variables X and Y respectively.

$$\text{MMD}[\mathcal{F}, P, Q] := \sup_{f \in \mathcal{F}} \left(\frac{1}{m} \sum_{i=1}^{m} f(x_i) - \frac{1}{n} \sum_{i=1}^{n} f(y_i) \right) \tag{19}$$

The MMD given by equation 19 is however unusable without specification of the class of functions, f. From earlier we know $\mathcal{F}$ is the set of functions f in the unit ball of RKHS, that is $\mathcal{F} = \{f \in \mathcal{H} \mid \|f\|_{\mathcal{H}} \leq 1\}$. A feature mapping $\phi(X)$ exists from $\mathcal{X}$ to $\mathbb{R}$ such that $f(X) = \langle f, \phi(X) \rangle_{\mathcal{H}}$. This feature mapping takes the canonical form $\phi(X) = k(X, \cdot)$, where $k(X, \cdot)$ is a kernel function, $k(x_1, x_2) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, centred at X with one fixed and free parameter respectively. For $k(X, \cdot)$ to be a valid kernel it must satisfy the following properties, **Reproducing Property**: $f(x) = \langle f, k(X, \cdot) \rangle_{\mathcal{H}} \ ^{\forall} \ f \in \mathcal{F}$, **Symmetry**: $k(x_1, x_2) = k(x_2, x_1)$, **Positive Definiteness**: meaning for any finite set of points e.g. $\mathbf{x}$ or $\mathbf{y}$, the gram matrix $K = [k(x_i, x_j)]_{i,j=1}^{n} \geq 0$, i.e. K is positive semi-definite, **Continuity** and **Universality**.

So evaluating f at X is equivalent to projecting f onto $\phi(X)$, or evaluating the functions at X is the same as taking the inner product between the entire function with a kernel function in an RKHS.

$$\begin{aligned} \mathbb{E}_X[f(X)] &= \mathbb{E}_X[\langle f, \phi(X) \rangle] \\ &= \langle f, \mathbb{E}_X[\phi(X)] \rangle_{\mathcal{H}} \\ &= \langle f, \mathbb{E}_X[k(X, \cdot)] \rangle_{\mathcal{H}} \\ &= \langle f, \mu_p \rangle_{\mathcal{H}} \end{aligned}$$

$\mathbb{E}_Y[f(Y)]$ follows by symmetry, substituting in 19 and following from Lemma 4 and Lemma 6 [20] we have ...

$$\begin{aligned} \text{MMD}^2[\mathcal{F}, P, Q] &= \|\mu_P - \mu_Q\|_{\mathcal{H}}^2 \\ &= \mathbb{E}_{X,X'}[k(X, X')] - 2\mathbb{E}_{X,Y}[k(X, Y)] + \mathbb{E}_{Y,Y'}[k(Y, Y')] \end{aligned} \tag{20}$$

Where X' and Y' are independent copies of random variables X and Y with equal distributions. Following these Lemmas the MMD_u^2 reduces to the distance between mean embeddings. The kernel trick allows us to expand the squared MMD in kernel terms, requiring no explicit computation in a infinite-dimensional Hilbert space, $\langle \mu_P, \mu_Q \rangle = \mathbb{E}_{X,Y}[\langle \phi(X), \phi(Y) \rangle] = \mathbb{E}_{X,Y}[k(x, y)]$.

Assuming that we have equal-sized sets of observations of our random variables, $m = n$, the unbiased empirical estimate of the MMD^2 is ...

Definition 7: The empirical unbiased maximum mean discrepancy squared.

$$\mathrm{MMD}_u^2[\mathcal{F}, P, Q] := \frac{1}{m(m-1)} \sum_{i \neq j}^{m} h(z_i, z_j) \tag{21}$$

Where $h(z_i, z_j) := k(x_i, x_j) + k(y_i, y_j) - k(x_i, y_i) - k(x_j, y_i)$

21 will form our MMD based objective function. It is worth noting if X and Y have identical probability measures, $P = Q$, then $\mathrm{MMD}_u^2 = 0$ (A requirement of a metric).

3.3.3 Wasserstein Distance

The last of our suggested objective functions for our model is the Wasserstein distance. The Wasserstein set of functions are metrics on probability distributions that measures the minimal effort required to reconfigure the probability mass of one distribution in order to recover the other distribution [21]. The Wasserstein distance overcomes some drawbacks of other metrics, for example the Wasserstein distance does not ignore the underlying geometry of the space unlike the L^p distance, allowing us to factor in the proximity of two distributions.

In our simple model, we take only asset returns as input, and therefore we consider the Wasserstein distance for $d = 1$, in the continuous case using the same notation as in MMD for p, X, Y, P and Q.

$$W_p(P, Q) = \left(\int_0^1 |F^{-1}(z) - G^{-1}(z)|^p \right)^{\frac{1}{p}} \tag{22}$$

Where F and G are the respective anti-derivatives or CDF functions of densities P and Q, the inverse of the CDF's correspond to the quantile of the distribution. The empirical equivalent of 22 is ...

Definition 8: The empirical one-dimensional Wasserstein distance.

$$W_p(P, Q) = \left(\sum_{i=1}^{n} \|X_{(i)} - Y_{(i)}\|^p \right)^{\frac{1}{p}} \tag{23}$$

Where $X_{(1)} \leq X_{(2)} \leq \ldots X_{(n)}$ and $Y_{(1)} \leq Y_{(2)} \leq \ldots Y_{(n)}$. This may look like the Euclidean distance, although it is different. The subscripts in brackets denote that X and Y are order statistics. This means unlike the Euclidean, L^2, distance the Wasserstein distance is invariant under permutation.

3.4 Data Collection

We source our data from Kaggle [22]. Our data contains a random list of 30 large-cap US-traded stocks from 2013-2023. We trim our data for consistency and usability to account for missing values and non available data as some stocks has more recent IOP's. Our final data-frame has 1579 rows and 30 columns. To get a better idea of the structure of observations in our dataset, we plot a combined histogram of each asset's returns [Figure 16]. We use a $70\% - 30\%$ split of training data to validation data. The validation test data is used to estimate the out of sample performance of our models in comparison to our benchmarks (Chapter 4.6)

4 Model Ouput

4.0.1 Training Model Output

For the last layer, l, in our network, $l = L$ our model will output some predicted weight vector, $\hat{\mathbf{w}}_t \in \mathbb{R}^n$. The subscript of time t here indicates that this is the predicted weight vector corresponding to the t'th asset return vector of observations. Our "optimal" return vector, obtained post-optimisation, is denoted as $\hat{\mathbf{w}}^*$.

To yield the predicted portfolio return for the i'th input vector we take the dot product between predicted weights $\hat{\mathbf{w}}_t$ and returns, $\mathbf{R}_t = (R_{1,t}, ..., R_{n,t})^{\ \forall}\ t = (1, \ldots, T)$, $R_i \in \mathbb{R}^T, \mathbf{R}_t \in \mathbb{R}^n$ ...

$$\begin{aligned} \hat{R}_t &= (z^{[L]}) \cdot z^{[0]} \\ &= \hat{\mathbf{w}}_t^\top \cdot \mathbf{R}_t \end{aligned} \tag{24}$$

Where $\hat{R}_t \in \mathbb{R}$. $z^{[L]}$ is the last model activation layer. $z^{[0]}$ is the first layer pre-activation (0 indexed) equal to the model input, $\mathbf{R}_t$, which is a vector of returns of n assets at time t.

Equation 24 is however a biased model prediction, this is because transaction costs are not factored in. A Transaction cost is a cost incurred from the investor when completing the financing a trade. These market frictions must be factored into our models for more accurate simulation of live-trading performance. Examples of transactions costs include price slippage, broker fees, bid-ask spreads or rebalancing costs. Another limitation of the return modelling in 24 is it does not factor asset volatility, meaning that assets with equal weights can contribute very differently to returns, with the most volatile asset dominating portfolio risk.

We implement a similar method to [10] by modifying predicted returns to factor in volatility scaling and transaction costs. In-sample predictions, however, will not factor in transaction costs as no buying and selling is actually happening, so there is no need to simulate these market dynamics. Model training returns, $\hat{R}$ are modelled differently from out-of-sample portfolio returns $\hat{R}_P$. In the in-sample case.

$$\begin{aligned} \hat{R}_i &= \sum_{j=1}^{n} \hat{w}_j \cdot R_{j,t} \cdot \frac{\sigma_{trg}}{\sigma_{j,t-1}} \\ &= \hat{\mathbf{w}}_t^\top \cdot \mathbf{R}_t \odot \boldsymbol{\sigma} \end{aligned} \tag{25}$$

Where $\odot$ defines an element-wise product, σ_{trg} is our target volatility form our target distribution and $\sigma_{j,t-1}$ is the ex-ante volatility of asset i calculated using an exponential weighted standard deviation 50 window moving average [10]. Modelling of out-of-sample portfolio returns will be discussed in Chapter 4.5.2.

4.0.2 Model Predicted Output

Our trained model is expected to have learned the "optimal" weight vectors that result in the least distance between predicted returns, $\hat{R}$, and target returns R^*. At first to obtain the optimal return vector, $\hat{\mathbf{w}}^* \in \mathbb{R}^n$, we averaged the asset weight vectors obtained from the optimised model. To then calculate the out-of-sample returns of buying and holding this portfolio at any time t, we take the dot product between $\hat{\mathbf{w}}^*$ and R_{unseen}, the result is our **predicted out-of-sample returns** R_P.

However to better align our methods with machine learning general practice we decided to use the optimised model out of sample to predict optimal weights $\hat{\mathbf{w}}^*$. This method also allows us to better judge our model out of sample performance, it also allows us to achieve

dynamic portfolio construction as the network will output a new set of weights for each new set of input data.

All performance metrics rely on the calculation of out-of-sample portfolio returns, which are adjusted for rebalancing and transaction costs. Portfolio rebalancing is the process of changing a portfolio's weight allocation to ensure a constant target weight or is maintained or to communicate a new set of preferences. Given that we rebalance of weights daily, our out-of-sample performance makes the assumption of end of day rebalancing. By not including any associated costs, we are effectively simulating frictionless market settings which do not provide the most accurate setting to calculate out-of-sample performance. We will factor market friction in our using equation 7 in [10].

$$\hat{R}_{P,t} = \sum_{i=1}^{n} \frac{\sigma_{tgt}}{\sigma_{i,t-1}} \hat{\mathbf{w}}_{i,t}^{*} \cdot r_{i,t} - C \cdot \sum_{i=1}^{n} \left| \frac{\sigma_{tgt}}{\sigma_{i,t}} \hat{\mathbf{w}}_{t}^{*} - \frac{\sigma_{tgt}}{\sigma_{i,t-1}} \hat{\mathbf{w}}_{i,t-1}^{*} \right| \tag{26}$$

Where Equation 26 specifies the portfolio out of sample returns, $\hat{R}_P$. $\hat{\mathbf{w}}_{i,t}^{*}$ is the optimal portfolio weight of the ith asset at time t. σ_{tgt} is the volatility target that we use to scale our position and insure a unfirm contribution to volatility across all our weighted assets. If we make an assumption of frictionless markets, C goes to 0 so the second term in 26 drops out, simplifying computation of out of sample returns. This assumption is unrealistic, as it would imply we can buy and sell assets without incurring any transaction costs, which is impossible in even the most liquid of markets. All out-of-sample model predictions will factor in transaction costs, $C \geq 0$.

4.0.3 Volatility Scaling

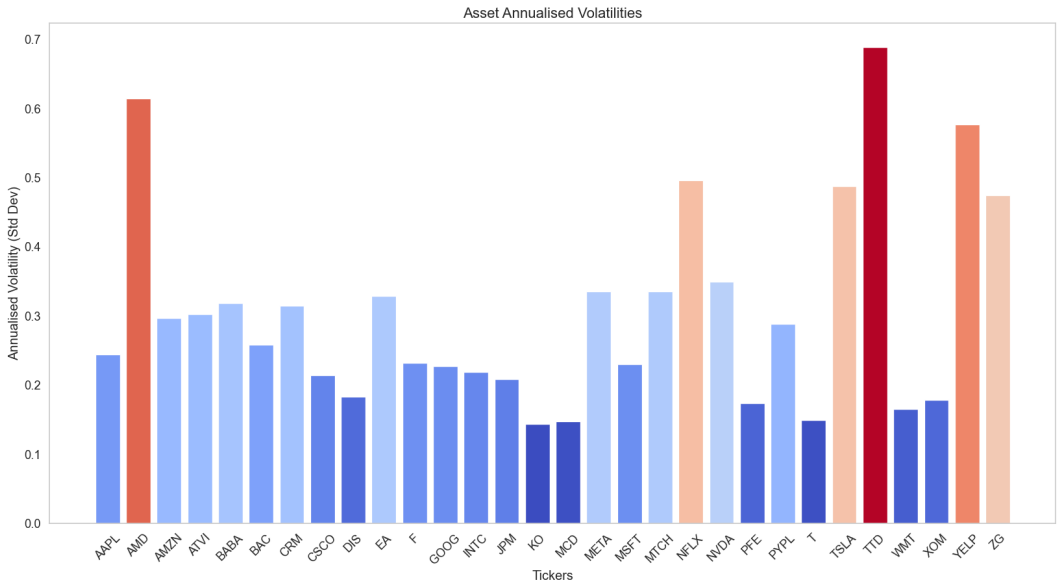

Figure 4: Out-of-sample asset return volatility. All volatilities are calculated using the unbiased anualised standard deviation.

Figure 4 shows how different assets in our portfolio have different volatilities. This represents the importance of volatility scaling as assets with higher volatilities can contribute more to portfolio returns even if the assets have identical weights. To mitigate the effect of dominating volatility, we adjust weights by normalisng volatilities.

We calculate the moving average of the standard deviation of returns using the exponential moving average (EMA) as in [10].

$$\text{EMA}_{var,t}(R_t) = \alpha \cdot \sigma_{i,t} - (1-\alpha)\text{EMA}_{var,t-1}(R_t) \tag{27}$$

Where $\alpha = \frac{2}{\text{window}+1}$ is the half-life of the or "memory" of the (EMA). Where the weight of the observation drops to half its value after $\frac{\text{window}}{2}$ period. The current volatility is $\sigma_{i,t} = \sqrt{\text{Var}(R_t)} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(R_t - \bar{R_t})^2}$ and $\sigma_{i,t-1}$ is the ex-ante volatility.

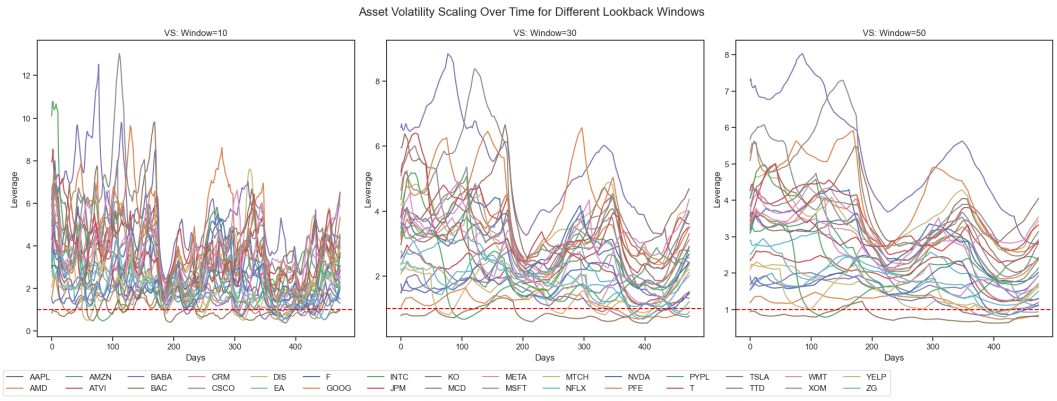

Figure 5: Exponential moving average of unbiased standard deviation of returns (not annaulised), for different window lengths in ascending order (10-50).

Figure 5 shows how using volatility scaling is identical to holding a leveraged position in the asset (y-axis). Increasing the window has the effect of more stable long-term volatility estimates and a lower leveraged position in the asset. The red dash line is the zero leverage line, at this level the asset volatility is equal to the target volatility, meaning the volatility scaling and leverage is zero. Assets below this line are more volatile than our target and have to be divested. A majority of assets are, however, above the zero leverage line, meaning they are less volatile than our target and must be levered to achieve the target volatility.

4.0.4 Market Position & Activation Functions

The calculation of our predicted weights vector is dependent on whether we define our portfolio as long only or long-short. When we are long an asset, we are making a bet that the price will increase. We can make this bet by buying the underlying. Being long, therefore, requires all weights to be non-negative as we assign a positive unit of wealth to the asset. Shorting allows us to bet on the asset price decreasing. This entails borrowing the asset (sell it) from a lender, and returning it (buying it back) at a later date. Long/short portfolios thus comprise of positive and negative weights. We impose one last constraint, that our wealth is fully invested, which means the absolute value of all weights must sum to one. Although we outline how portfolio weights will be calculated in our model for either case, we will not present results for the long-only case. This is largely because the results we obtained are not significantly different.

To impose a unit invested amount for the Long/Short case, we must normalise $\text{Tanh}(\hat{w}_{i,t}) \in [-1, 1]$ by the sum $\text{Tanh}(\hat{\mathbf{w}}_t)$.

$$\hat{w}_{i,t} = \frac{\text{Tanh}(\hat{w}_{i,t})}{\sum_{j=1}^{n}|\text{Tanh}(\hat{w}_{j,t})|} \tag{28}$$

Portfolio Type	Activation Function	Formula
Long Only	Softmax(x_i)	$\frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$
Long/Short	Tanh(x)	$\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

Table 1: Activation functions used for different portfolio types. The softmax function ensures long-only portfolios while the hyperbolic tangent function (tanh) allows for long/short positions.

4.1 Differentiable Objectives

The objective/loss function of our models must be differentiable to insure we can optimise our model's parameters and learn weights that best fit our target distribution. Derivatives of all functions below will be taken with respect to the directly preceding layer in our network, and inputs are identical across all loss functions. It is important to note that PyTorch's `.backward()` class method will calculate all these derivatives automatically, no need for manual definition, including these results is largely for completeness and to communicate a better explanation of our method.

4.1.1 Loss Function Inputs

As stated prior all loss functions will take equal inputs. These are a variable $\hat{R}_i \in \mathbb{R}$ and a constant $R_i^* \in \mathbb{R}$ which is the i'th sampled target portfolio return. To simulate $R^* \in \mathbb{R}^T$ we can draw samples from a known distribution with predefined parameters; the set of such samples define our discrete target distribution. The portfolio manager would define a desirable target distribution that they wish to replicate. For example, suppose a portfolio manager wishes to construct a momentum-driven portfolio; to reflect these preferences, they may wish to define a target distribution skewed to the right-hand side, as larger positive returns relative to negative returns indicate the portfolio captures long-lasting positive moves. The model will then attempt to fit to the distribution by learning weights that correspond to momentum-driven assets. Two assumptions are made here ...

- **Univariate Representation of Preferences.** The portfolio manager's preference can be modelled using a univariate distribution.
- **Perfect Reflection of Preferences.** The portfolio manager must be able to reflect their preferences through a distribution and its parameters perfectly.

4.1.2 L_2^2 Loss Function Derivative

$$\frac{d\ d(\hat{R}_i, R^*)_{L_2^2}}{d\hat{R}} = \frac{d}{d\hat{R}} \left[\|\hat{p}(\hat{R}) - \hat{q}(R_p)\|_2^2 \right] \tag{29}$$

Guassian kernel functions are discrete approximations of the PDF, i.e. the PMF functions. Solving using chain rule we get ...

$$\begin{aligned} \frac{d}{d\hat{R}} \left[\|\hat{p}(\hat{R}) - \hat{q}(R^*)\|^2 \right] &= 2 \left[\hat{p}(\hat{R}) - \hat{q}(R^*) \right] \cdot \left(-\hat{R} \cdot \hat{p}(\hat{R}) \right) \\ &= -2\hat{R} \cdot \hat{p}(\hat{R}) \left[\hat{p}(\hat{R}) - \hat{q}(R^*) \right] \end{aligned} \tag{30}$$

4.1.3 Empirical MMD_u^2 Loss Function Derivative

$$\begin{aligned}\frac{d\ d(\hat{R}, R^*)_{\text{MMD}_u^2}}{d\hat{R}_i} &= \frac{d}{d\hat{R}_i}\Bigg[\frac{1}{n(n-1)}\sum_{i=1}^{T}\sum_{j\neq i}k(\hat{R}_i,\hat{R}_j) + \frac{1}{n(n-1)}\sum_{i=1}^{T}\sum_{j\neq i}k(R_{i,trg},R_j^*) + \\ &\quad \frac{2}{n^2}\sum_{i=1}^{T}\sum_{j=1}k(\hat{R}_i,R_j^*) \qquad (31)\\ &= \frac{2}{n(n-1)}\sum_{j\neq i}^{T}\nabla_{\hat{R}_i}k(\hat{R}_i,\hat{R}_j) - \frac{2}{n^2}\sum_{j=1}^{T}\nabla_{\hat{R}_i}k(\hat{R}_i,R_j^*)\end{aligned}$$

$$\begin{aligned}\implies \frac{d\ d(\hat{R}, R^*)_{\text{MMD}_u^2}}{d\hat{R}} &= (a)^\top\left[\mathbf{1}^\top \text{J}_{\hat{R}}(k(\hat{R},\hat{R}))\mathbf{1} - \text{tr}(\text{J}_{\hat{R}}(k(\hat{R},\hat{R})))\right] - (b)^\top\left[\mathbf{1}^\top \text{J}_{\hat{R}}(k(\hat{R},R^*))\mathbf{1}\right]\\ &= (a)^\top\left[\mathbf{1}^\top \text{J}_{\hat{R}}(K)\mathbf{1} - \text{tr}(\text{J}_{\hat{R}}(K))\right] - (b)^\top\left[\mathbf{1}^\top \text{J}_{\hat{R}}(K')\mathbf{1}\right] \qquad (32)\end{aligned}$$

Where $\sum_{i,j}^{n} k(\hat{R}_i,\hat{R}_j) = K \in \mathbb{R}^{T\times T}$, $\sum_{i=1}^{n}\sum_{j\neq i} k(\hat{R}_i,\hat{R}_j) = K - \text{tr}(K)$, $\sum_{i,j}^{n} k(\hat{R}_i,R_j^*) = K' \in \mathbb{R}^{T\times T}$ and $\mathbf{1} \in \mathbb{R}^T$

4.1.4 Empirical One-Dimensional Wassertein Loss Function Derivative

$$\frac{d\ d(\hat{R}, R^*)_{W_{p=2}}}{d\hat{R}} = \frac{\hat{R}_{(i)} - R^*_{(i)}}{\|\hat{R}_{(i)} - R^*_{(i)}\|_2} \qquad (33)$$

Which is similar to the result in **??** if the L_2^2 divergence would reduce to the L^2 distance. The only difference is in the inputs, which are order statistics rather than unordered observations.

4.2 Model

In this section, we attempt to clearly explain our model's architecture. We will also discuss a performance method for measuring the "goodness of fit" of our DmNN model predictions to the target, i.e. how well does DmNNs predicted model portfolio returns fit to the target distribution of returns.

4.2.1 Distance Metric Neural Network (DmNN) Model Architecture

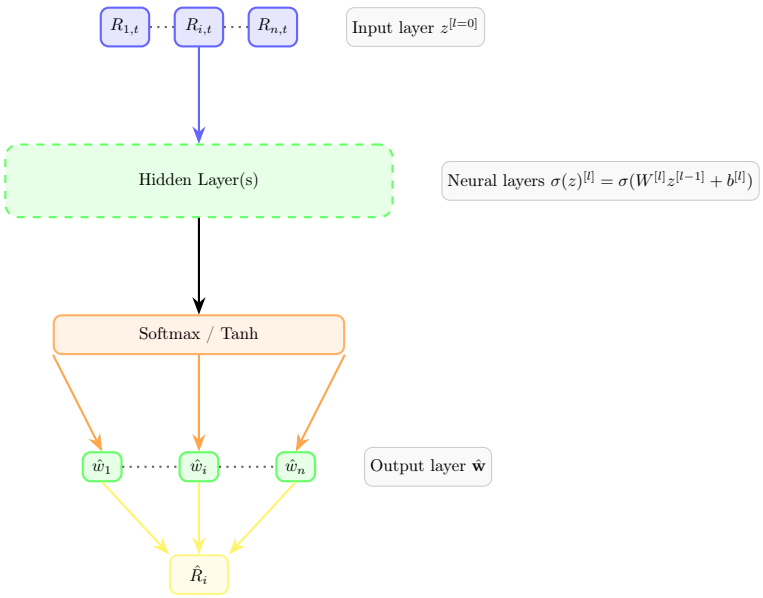

Figure 6: DmNN model architecture schematic (To represent the constructuction of long-short portfolios we would swap Softmax with normalised Tanh(x) [28]). Our model is a simple MLP that contains three main components: an input layer, a hidden layer, and an output layer.

The model in [Figure 6] is a fully connected neural network (FCN), specifically, our model is referred to as a multilayer perceptron (MLP).

Input Layer. Our MLP takes the vector $\mathbf{R}_t$ as input, this is the t'th observation of n asset returns. Note we use simple returns as those defined for [2]. The number of inputs into our model is therefore equal to the number of assets.

Hidden Layer. These are the intermediate layers in our MLP. We define one hidden layer, such that we have a network depth of 4 layers (omitting the input layer). The reason for this is to maintain simplicity and explainability. Our network contains no more than 200 nodes at any layer.

Output Layer. We apply activation functions as specified in Section 4.0.4 to the pre-activation in layer L such that we get weight values whose absolute value sum to 1, to ensure we meet our full investment of wealth constraint.

4.2.2 Distance Metric Long Short-Term Memory (DmLSTM) Model Architecture

The second of our neural network models is an LSTM 2.5, DmLSTM. It would be interesting to compare the performance of plain vanilla models (DmNN) to more complicated models using the same loss functions, to see if more complicated models translate to superior performance.

We select LSTMs as a candidate model due to their ability to extract temporal dependencies across time series data (return data). We define our DmLSTM with two recurrent layers with the last layer outputting our weight estimates, We also give DmLSTM a hidden

state and cell state size of 100 feature observations. The input and output dimensions of DmLSTM are unchanged from DmNN. The model architecture is specified below.

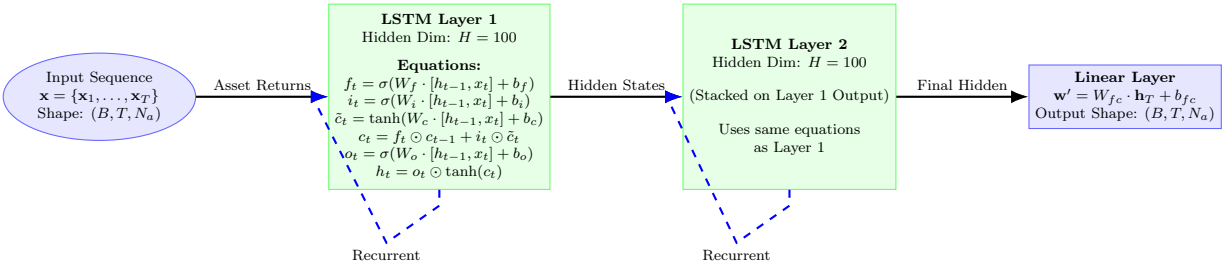

Figure 7: DmLSTM model architecture. The LSTM has five main components a hidden state, cell state and three gate types: forget gate, input gate and output gate. B represents the batch size, here 1, T is the sequence length of observatins here m and N_a is the number of assets n.

Following from the pre activation "Linear Layer" in Figure 7, this would be passed through the last layer activation function to extract the set of portfolio weights corresponding to a long or long-short portfolio.

4.3 A Walk Through Model Evaluation

In this sub-chapter, we provide a high-level overview of how our proposed models are trained and are used to output dynamic portfolio weight predictions.

1. **Feedforward.** We pass all our asset returns $\mathbf{R}_t$ through the network and yield a weight vector $\hat{\mathbf{w}}_t$ for each set of observations (effectively a sample from a d d dimensional vector where the dimension is our number of assets) t. We do this for all $t = 1, \ldots T$.

2. **Backpropegation.** We now have T guesses of portfolio weights that satisfy our full investment constraint. At this stage, we have not made an attempt to better train our model to learn a specific task, which is required for a useful model, so we will attempt to find $f_{\hat{\theta}}$. As outlined earlier, we will use distance metrics in-sample as our loss functions (only for in-sample training). Our goal is to find the best model weights that map returns to some desirable distributions that reflect our risk preference.

3. **Validation.** We continue with the standard ML training pipeline. When it comes to prediction, we feed forward our out-of-sample data as input into our theoretically optimal trained model $f_{\hat{\theta}}(\mathbf{X})$, and then compute the error of these predictions. If the error is low, it means the model generalises well on unseen data, making it a suitable candidate for predictions.

4.4 Discrete Hypothesis Testing

Our distance metrics form our test statistic. We can conduct a hypothesis test comparing if our observed value of a test statistic $\hat{d}(\hat{R}_t, R_{t,trg})$ belongs to the null distribution. However, the distribution of our test statistic is unknown, meaning we cannot compare the $p - value$ to the α level test, where the observed $p - value$ is $p(d) = 1 - F(\hat{p})$, and F is the cumulative distribution function (CDF). To carry out the hypothesis test we rely on non-parametric estimators such as the **bootstrap data generating process** (bootstrap DGP) to form the **empirical distribution function** (EDF).

Given our distance functions are valid metrics, significantly small observed values are likely to indicate that our model has fitted the distributions well. We therefore have a left-tailed test and our $p - value$ from the bootstrap DGP is [23] ...

$$\hat{p}^*(\hat{d}) = \frac{1}{B}\sum_{i=1}^{B} \mathbb{I}\left[|d_j^*| \geq |\hat{d}|\right] \tag{34}$$

Where $\hat{p}^*(\hat{d})$ is the estimated bootstrap-generated $p - value$, the hat indicates we have an estimate, and the superscript star indicates that the estimate was generated from the bootstrap DGP. d_j^* is the j'th simulated test statistic.

We reject the null when $\hat{p}^*(\hat{d}) < \alpha$ which is equivalent to rejecting he null if $\hat{d} < \hat{F}_\alpha^*(d^*)$ where d^* is the α quantile corresponding to the bootstrap generated $p - value$.

The hypothesis test we conduct is to test $\boldsymbol{H_0}$: *"distributions are equal"* against $\boldsymbol{H_1}$: "*distributions are different"*. We wish to retain the null hypothesis, therefore we want sufficiently large $p - values$ that are greater than our significance level. To carry out a hypothesis test we start by calculating an observed distance value post model optimisation, this is our $\hat{d}$, and tells us how similar out model predictions are to the target. To form our null distribution we make an assumption both predicted and target distributions are equal, so we combine our samples and compute the metric of choice (a distance metric, the same as $\hat{d}$, from permuted samples from this set. We expect to retain H_0, then it is expected we observe many simulated samples (metrics estimated from the pooled dataset of returns) which are less extreme than our observed metric value and hence a large $p - value > \alpha$ is required to retain $\boldsymbol{H_0}$.

4.5 Model Algorithms

In this sub-chapter, we provide the mathematically based pseudocode for training our proposed models. Optimisation/training of our models is done using the Adam optimisation scheme [24], although the pseudocode for the respective backward pass algorithms will not be provided. We implement all models in Python using the PyTorch package. All models and code used to generate results discussed in this paper can be found in this paper's respective GitHub repository [25].

4.5.1 DmNN Forward Pass

Algorithm 3 Distance Metric Neural Network (DmNN) Model Forward Pass

Require: Input dimension d_{in}, output dimension d_{out}, hidden layers $H = 2$, neurons per layer $n = 100$, long-short flag ℓ
Ensure: Portfolio weights $\mathbf{w} \in \mathbb{R}^{d_{out}}$

Initialize:
  Input layer: $\mathbf{W}^{(1)} \in \mathbb{R}^{d_{in} \times n}, \mathbf{b}^{(1)} \in \mathbb{R}^n$
  Hidden layers: $\mathbf{W}^{(h)} \in \mathbb{R}^{n \times n}, \mathbf{b}^{(h)} \in \mathbb{R}^n$ for $h = 2, \ldots, H$
  Output layer: $\mathbf{W}^{(H+1)} \in \mathbb{R}^{n \times d_{out}}, \mathbf{b}^{(H+1)} \in \mathbb{R}^{d_{out}}$
  Activation function: $\sigma(\cdot) = \begin{cases} \tanh(\cdot) & \text{if } \ell = \text{True} \\ \text{softmax}(\cdot) & \text{if } \ell = \text{False} \end{cases}$
procedure FORWARD($\mathbf{x}$)
  Input: $\mathbf{x} \in \mathbb{R}^{B \times d_{in}}$ (batch of feature vectors)
  Forward Propagation:
  $\mathbf{a}^{(1)} \leftarrow \mathbf{x}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$ ▷ Input layer
  $\mathbf{z}^{(1)} \leftarrow \text{ReLU}(\mathbf{a}^{(1)})$ ▷ $\text{ReLU}(a) = \max(0, a)$
  for $h = 2$ **to** H **do**
    $\mathbf{a}^{(h)} \leftarrow \mathbf{z}^{(h-1)}\mathbf{W}^{(h)} + \mathbf{b}^{(h)}$ ▷ Hidden layer h
    $\mathbf{z}^{(h)} \leftarrow \text{ReLU}(\mathbf{a}^{(h)})$
  $\mathbf{r} \leftarrow \mathbf{z}^{(H)}\mathbf{W}^{(H+1)} + \mathbf{b}^{(H+1)}$ ▷ Raw weights: $B \times d_{out}$
  Weight Normalization:
  if $\ell = \text{True}$ (Long-Short Strategy) **then**
    $\mathbf{s} \leftarrow \tanh(\mathbf{r})$ ▷ Apply hyperbolic tangent
    $\mathbf{d} \leftarrow \sum_{j=1}^{d_{out}} |\mathbf{s}_{:,j}|$ ▷ Sum of absolute values per sample
    $\mathbf{w} \leftarrow \frac{\mathbf{s}}{\mathbf{d}+\epsilon}$ ▷ L1-normalize, $\epsilon = 10^{-8}$
  else ▷ Long-Only Strategy
    $\mathbf{w} \leftarrow \text{softmax}(\mathbf{r})$ ▷ Probability simplex projection
  return $\mathbf{w} \in \mathbb{R}^{B \times d_{out}}$ ▷ Portfolio weights

Network Architecture:
  • Input layer: $\mathbb{R}^{d_{in}} \to \mathbb{R}^n$ with ReLU activation
  • Hidden layers: $\mathbb{R}^n \to \mathbb{R}^n$ with ReLU activation (repeated $H - 1$ times)
  • Output layer: $\mathbb{R}^n \to \mathbb{R}^{d_{out}}$ with constraint-specific activation

Portfolio Constraints:
  • Long-short: $\sum_{j=1}^{d_{out}} |w_j| = 1$, $w_j \in [-1, 1]$
  • Long-only: $\sum_{j=1}^{d_{out}} w_j = 1$, $w_j \geq 0$

4.5.2 DmLSTM Forward Pass

Algorithm 4 Distance Metric Long-Short-Term-Memory (DmLSTM) Model Forward Pass

Require: Input dimension d_{in}, output dimension d_{out}, hidden dimension $d_h = 100$, number of layers $L = 2$, long-short flag ℓ
Ensure: Portfolio weights $\mathbf{w}_t \in \mathbb{R}^{d_{out}}$ for each time step t

Initialize:
  LSTM network: $\text{LSTM} : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_h}$ with L layers
  Linear layer: $\mathbf{W}_{fc} \in \mathbb{R}^{d_h \times d_{out}}, \mathbf{b}_{fc} \in \mathbb{R}^{d_{out}}$
  Activation function: $\sigma(\cdot) = \begin{cases} \tanh(\cdot) & \text{if } \ell = \text{True} \\ \text{softmax}(\cdot) & \text{if } \ell = \text{False} \end{cases}$
procedure FORWARD($\mathbf{X}, \mathbf{h}_0$)
  Input: $\mathbf{X} \in \mathbb{R}^{B \times T \times d_{in}}$ (batch of sequences)
  Input: $\mathbf{h}_0 \in \mathbb{R}^{L \times B \times d_h}$ (initial hidden states, optional)
  if $\mathbf{h}_0 = \text{None}$ **then**
    $\mathbf{H}, \mathbf{h}_T \leftarrow \text{LSTM}(\mathbf{X})$
  else
    $\mathbf{H}, \mathbf{h}_T \leftarrow \text{LSTM}(\mathbf{X}, \mathbf{h}_0)$
  $\mathbf{R} \leftarrow \mathbf{H}\mathbf{W}_{fc} + \mathbf{b}_{fc}$ ▷ Raw weights: $B \times T \times d_{out}$
  if $\ell = \text{True}$ (Long-Short Strategy) **then**
    $\mathbf{A} \leftarrow \tanh(\mathbf{R})$ ▷ Apply hyperbolic tangent
    $\mathbf{S} \leftarrow \sum_{i=1}^{d_{out}} |\mathbf{A}_{:,:,i}|$ ▷ Sum of absolute values along asset dimension
    $\mathbf{W} \leftarrow \frac{\mathbf{A}}{\mathbf{S}+\epsilon}$ ▷ Normalize by sum of absolute weights, $\epsilon = 10^{-8}$
  else ▷ Long-Only Strategy
    $\mathbf{W} \leftarrow \text{softmax}(\mathbf{R})$ ▷ Apply softmax normalization
  return $\mathbf{W} \in \mathbb{R}^{B \times T \times d_{out}}$ ▷ Portfolio weights

Key Properties:
  • Temporal dependency modeling via LSTM architecture
  • Causal processing: weights at time t depend only on returns up to $t-1$
  • Long-short constraint: $\sum_{i=1}^{d_{out}} |w_{t,i}| = 1$ when $\ell = \text{True}$
  • Long-only constraint: $\sum_{i=1}^{d_{out}} w_{t,i} = 1$, $w_{t,i} \geq 0$ when $\ell = \text{False}$

4.6 Benchmark Models

We have three benchmark models ...

- **MVO.** It is logical to compare our novel methods against original literature. In this case, we compare performance against the maximal Sharpe ratio portfolio from the mean-variance framework.
- **MVO via Black-Litterman Allocation (MVO+BL).** A novel improvement of the MVO framework.
- **DLS.** It is important to compare how our model performs against newer and inspired literature in the field, here the DLS framework outlined in [10].

4.7 Performance Metrics

These are the metrics we will use to standardise the performance of each model so that we can make model comparisons. All metrics are annualised by multiplying by a proportionality constant of the number of trading days in a year.

4.7.1 Sharpe Ratio

This is the return per unit risk, a risk-adjusted return metric.

$$\text{Sharpe Ratio} = \left(\frac{\mathbb{E}[R_P - r_f]}{\sqrt{\mathbb{E}[(R_P - \mathbb{E}[R_P])^2}} \right) \cdot \sqrt{252} \tag{35}$$

4.7.2 Sortino Ratio

This is a form of adjusted Sharpe ratio, although it takes into account the downside deviation of portfolio returns, it therefore takes into account the asymmetric nature of negative returns, using it directly in the model to better reflect risk.

$$\text{Sortino Ratio} = \left(\frac{\mathbb{E}[R_P - r_f]}{\mathbb{E}[(R_P - \mathbb{E}[R_P])^2 \mid \mathbb{E}[R_P < 0]]} \right) \cdot \sqrt{252} \tag{36}$$

4.7.3 Expected Return

This is the simple unbiased mean estimator.

$$\mathbb{E}[R_P] = \left(\frac{1}{T} \sum_{t=1}^{T} R_{t,P} \right) \cdot 252 \tag{37}$$

Given we compute this statistic for all our model runs across many portfolios, by the Central Limit Theorem, we expect the distribution of R_P to be normal with mean 0. That is via Lindeberg–Lévy CLT $\sqrt{T}(\mathbb{E}[R_P] - \bar{R}_P) \to^d N(0, \sigma^2)$.

4.7.4 Value at Risk (VaR)

This is the greatest lower bound of returns corresponding to the α percentile of returns, or in other words this is the value corresponding to the α case returns with $100 \cdot (1 - \alpha)\%$ confidence. We can interpret the VaR as the $100 \cdot (1 - \alpha)\%$ confidence with which returns will not exceed. Therefore the greater the magnitude of this value the more risky and hence worse performing the investment. For DmNN to be the best it must yield the lowest VaR. The VaR is

$$\text{VaR}_\alpha(R) = \min\{c : \mathbb{P}(R \leq c) \geq \alpha\} \tag{38}$$

Where R is the return and c is the threshold which we expect our return to be at least as extreme in the worst α of cases.

4.7.5 Conditional Value at Risk (CVaR)

A metric closely related to the VaR that measures the expected value of the losses that exceed the VaR threshold. CVaR is

$$\text{CVaR}_\alpha(R) = \mathbb{E}[R \mid R \geq \text{VaR}_\alpha(R)] \tag{39}$$

The CVaR gives the expected loss in the $(1 - \alpha)\%$ worst cases and can be interpreted as the average severity of the α quantile loss.

It is important to note that CVaR is a coherent risk measure, satisfying **Monotonicity**, **Sub-additivity**, **Translation Invariance**, and **Homogeneity**. VaR is not a coherent measure as it does not satisfy Sub-additivity.
The VaR and CVaR will both be computed for the validation set. This is the set of out-of-sample data that we use to evaluate our trained models, DmNN and DmLSTM, by comparing their performance against other competing methods (MVO, MVO+BL, DLS).

5 Results

5.1 Training Results

We train DmNN and DmLSTM varying only the distance metric and the use of volatility scaling. To decide on the number of times we would feed through our data in training to DmLSTM, we used the same number of runs as [10]. As for DmLSTM the choice of epochs was based of a test for the minimum number of runs required to fail to reject $\boldsymbol{H_0}$: "Densities are equal", we find the greatest lower bound of the number of epochs such that we fail to reject $\boldsymbol{H_0}$ to be 1000 epochs. This means that it takes 1000 epochs to observe a DmNN estimated portfolio return distribution that matches the target, which we determine by conducting a bootstrap DGP hypothesis test (Evidence of retaining $\boldsymbol{H_0}$ for the Wasserstein loss function, Figure 17).

To present how well our model learns the target distribution by mapping returns to weights, we plot the in-sample returns pre-training (generated from the first forward pass of all return data) and post-training against the empirical target return distribution. To form the target return distribution we sample realisations r_t from the random variable $R_t \sim N(\mu = 0.01, \sigma^2 = 0.016)$, where $t = (1, \ldots, m)$.

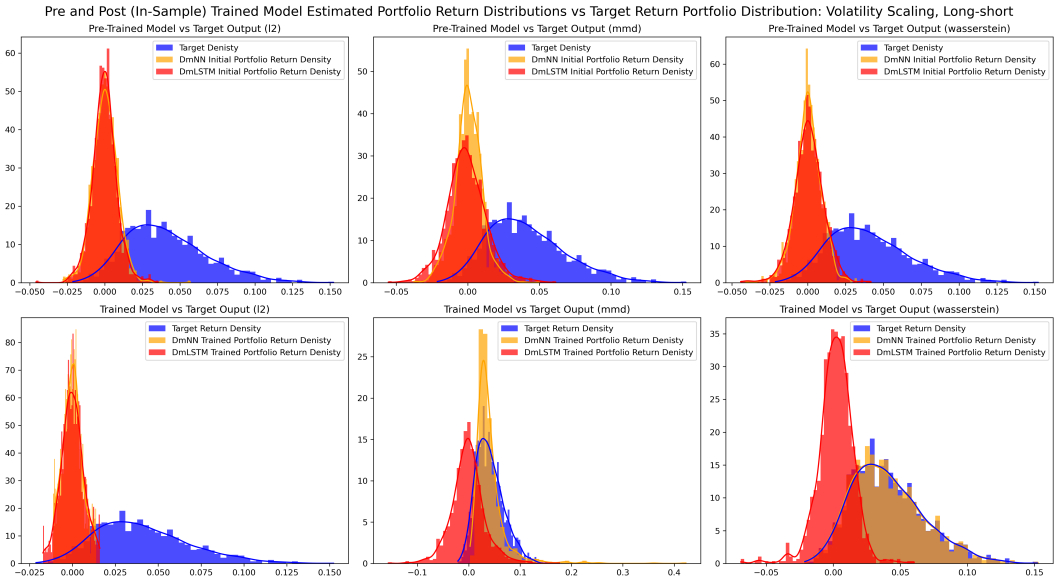

Figure 8: Initial model prediction (top) vs optimised model prediction (bottom) with **volatility scaling**, using L_2^2 distance, MMD_u^2 distance and W_2 distance

Figure 8 shows the empirical densities of model outputs (in-sample predicted returns) using volatility scaling [26] pre and post model training. All plots are overlaid with their continuous density approximations using the Gaussian kernel density estimator (KDE) [16].

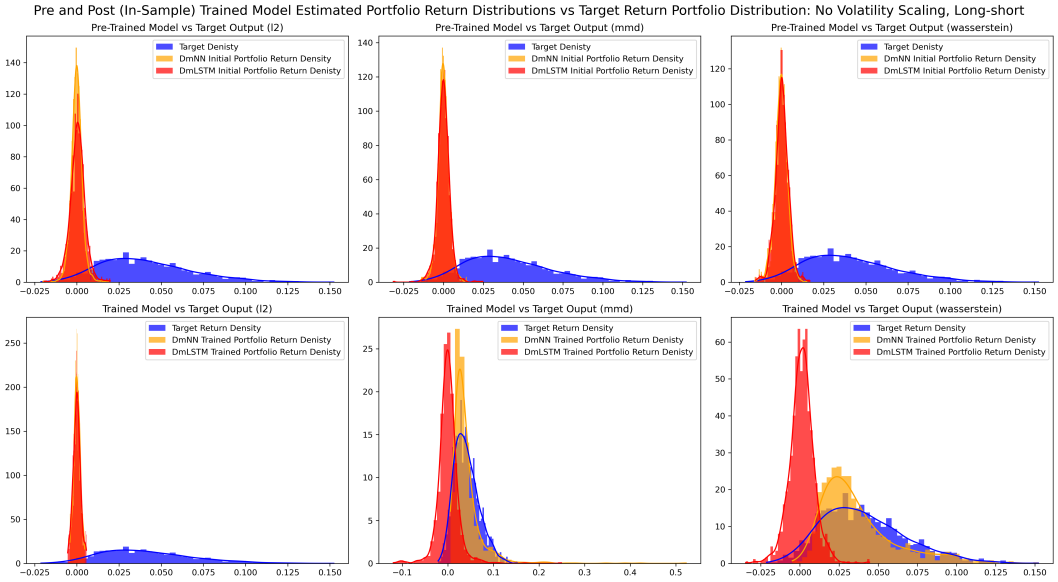

Figure 9: Initial model prediction vs optimised model prediction with **no volatility scaling**, using L_2^2 distance, MMD_u^2 distance and W_2 distance

Figure 9, is identical in interpretation to that in Figure 8, with the exception that our model does not use volatility scaling.

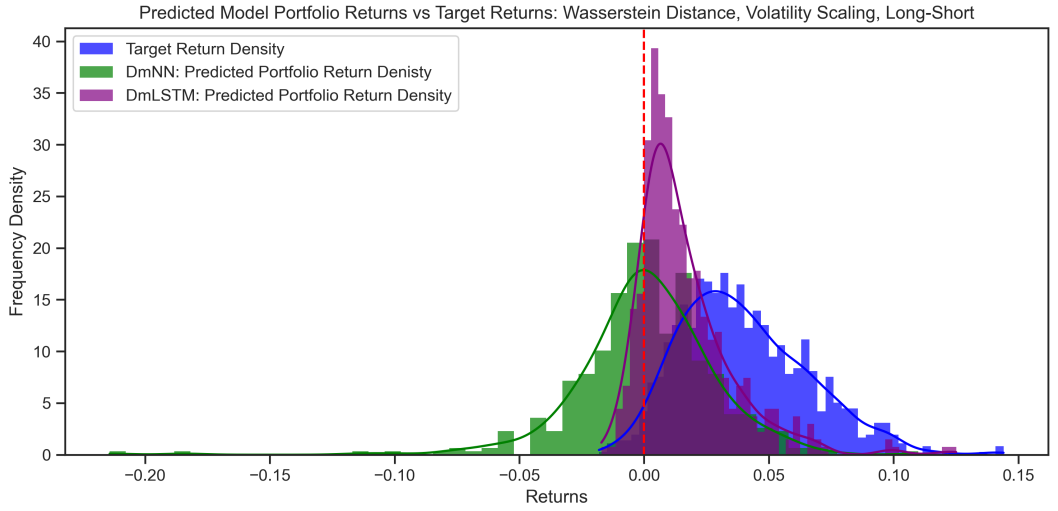

Figure 10: Empirical distribution estimates of **target distribution (blue)**, **optimised DmNN out-of-sample portfolio return distribution (green)** and **optimised DmLSTM out-of-sample portfolio return distribution (purple)**. Solid coloured lines are continuous density estimates using a Gaussian KDE.

Figure 10 shows how our models' out-of-sample estimated portfolio return densities compare to the target density.

5.2 Validation Model Results

All plots in this section are generated using the output of the model validation algorithm **??**. We also now consider a basket of 10 of 30 assets from which we construct $10C9 = 10$ unique datasets, each comprising of 9 assets (the number can be varied at the expense of longer computation time). We will train and validate our models on each one of these unique datasets. Our models (DmNN and DmLSTM) assume that return data are not independent, meaning our model fitted on some set of assets cannot be used to predict out-of-sample portfolio returns for a different set of assets.

5.2.1 Long-short

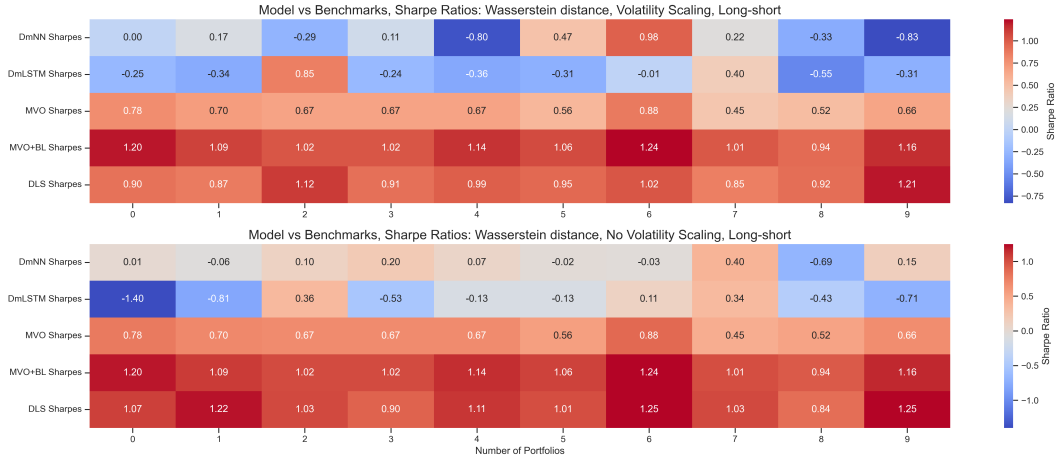

Figure 11: Long-short type models vs benchmark Sharpe ratios for 10 unique portfolios. **Top:** Models use volatility scaling, **Bottom:** No volatility scaling

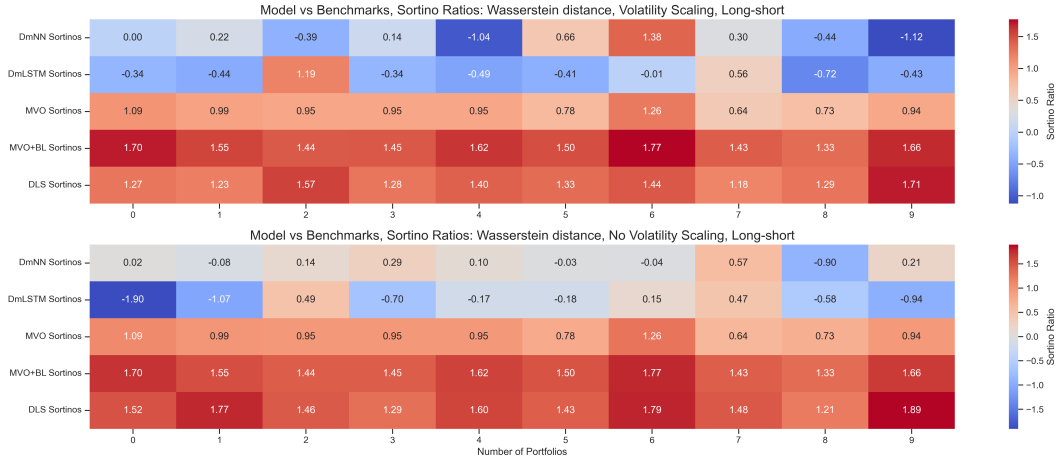

Figure 12: Long-short type models vs benchmark Sortino ratios for 10 unique portfolios. **Top:** Models use volatility scaling, **Bottom:** No volatility scaling

Figure 11 and Figure 12 show the out-of-sample annaulised Sharpe ratio and Sortino ratio performance of our volatility scaling and non-volatility scaling Long-short type models compared to the benchmark models.

5.2.2 Long-only

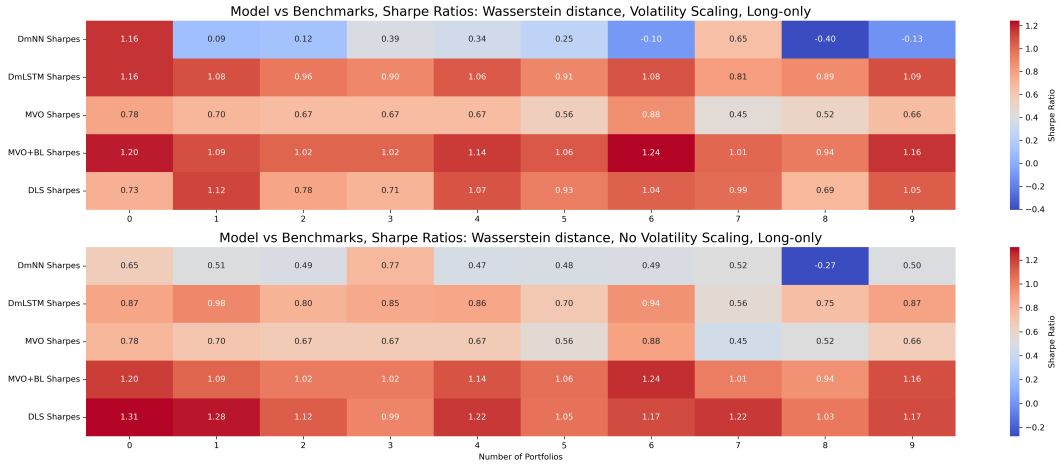

Figure 13: Long-only models vs benchmark annualised Sharpe ratios for 10 unique portfolios. **Top:** Models use volatility scaling, **Bottom:** No volatility scaling

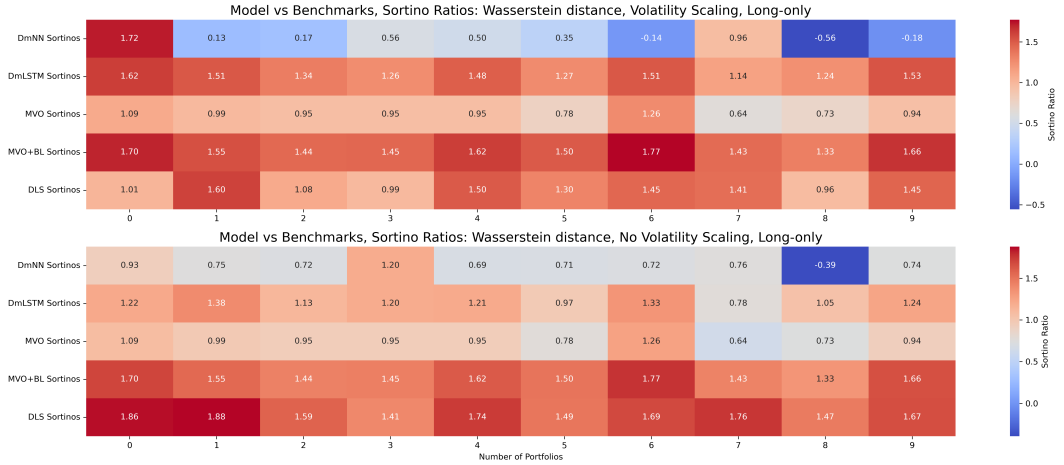

Figure 14: Long-only models vs benchmark annualised Sortino ratios for 10 unique portfolios. **Top:** Models use volatility scaling, **Bottom:** No volatility scaling

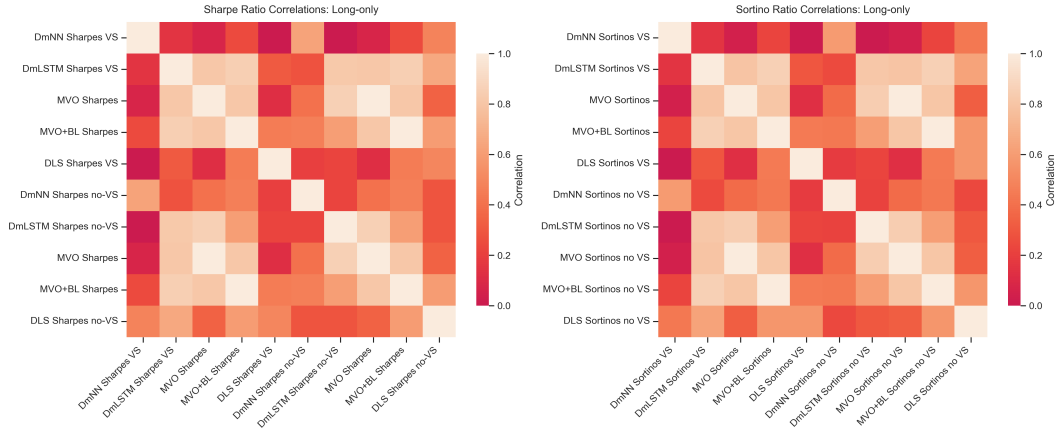

Figure 15: Long only, volatility scaling model's Sharpe ratio and Sortino ratio correlations

Figure 13 and Figure 14 show the out-of-sample annaulised Sharpe ratio and Sortino ratio performance of our volatility scaling and non-volatility scaling Long type models compared to the benchmark models. Figure 15 shows correlations between model Sharpe ratios and Sortino ratios for volatility scaling portfolios.

5.3 Risk Measure Results

VaR of Models and Benchmarks: Long-only, Volatility Scaling				
VaR DmNN	**VaR DmLSTM**	**VaR MVO**	**VaR MVO+BL**	**VaR DLS**
-6.13%	-4.51%	-2.25%	-1.93%	-5.55%
-6.97%	-4.71%	-2.41%	-2.16%	-5.76%
-6.98%	-4.08%	-2.18%	-2.01%	-5.18%
-6.47%	-4.43%	-2.18%	-2.00%	-5.25%
-6.96%	-4.60%	-2.18%	-2.20%	-4.97%
-7.20%	-5.65%	-2.58%	-2.45%	-5.56%
-7.05%	-4.15%	-1.89%	-1.89%	-5.30%
-6.22%	-4.27%	-1.98%	-2.12%	-4.87%
-6.95%	-4.44%	-2.00%	-1.87%	-5.55%
-6.97%	-4.94%	-2.20%	-2.40%	-4.97%

Table 2: Value at Risk (VaR) of predicted portfolio returns on out-of-sample data for long only DmNN and DmLSTM models with volatility scaling vs. Benchmarks, MVO, MVO+BL and DLS.

VaR of Models and Benchmarks: Long-only, No Volatility Scaling				
VaR DmNN	**VaR DmLSTM**	**VaR MVO**	**VaR MVO+BL**	**VaR DLS**
-4.81%	-1.70%	-2.25%	-1.93%	-1.91%
-4.83%	-1.94%	-2.41%	-2.16%	-2.45%
-4.81%	-1.76%	-2.18%	-2.01%	-2.01%
-3.73%	-1.72%	-2.18%	-1.99%	-2.00%
-4.81%	-1.75%	-2.18%	-2.20%	-2.20%
-4.84%	-1.98%	-2.58%	-2.45%	-2.64%
-4.83%	-1.73%	-1.89%	-1.89%	-1.86%
-4.70%	-1.68%	-1.98%	-2.12%	-2.20%
-3.38%	-1.84%	-2.00%	-1.87%	-1.82%
-4.80%	-1.84%	-2.20%	-2.40%	-2.28%

Table 3: Value at Risk (VaR) of predicted portfolio returns on out-of-sample data for long only DmNN and DmLSTM models with no volatility scaling vs. Benchmarks, MVO, MVO+BL and DLS.

6 Discussion

6.1 Return Distributions Pre and Post Training

With reference to Figures 8 and Figures 9 we see that trends among return densities are consistent across volatility scaling and non-volatility scaling models. The pre-trained model output, which is effectively one forward pass of the entire data through the model, poorly fits the target distribution. Both DmNN and DmLSTM produce similar predicted weights that result in near-normally distributed estimated portfolio return densities. This means that model outputs are invariant to model type and loss function specification pre-training.

Post-training (In-sample) only DmNN does a better job of assigning weights to each asset such that the portfolio's return distribution better approximates the target. Is it clear from the kernel density estimates of the PDF's that the Wasserstein metric does the best job of learning the target distribution, achieving very similar approximations post-training, and a near perfect fit in the volatility scaling case [Figure 8]. The L_2^2 divergence produces near identical results pre and post training across DmNN and DmLSTM models suggesting that it is a poor proxy for learning a distribution. The unbiased MMD^2 produces better fitting models when minimised, although only for DmNN. DmLSTM produces similar results pre and post-training, likely meaning it is a poor model for the portfolio distribution fitting task. The findings listed above are consistent among volatility scaling and non-volatility scaling specified models, we also find the same to be true when running the same tests for models with the long-only constraint.

Given the Wasserstein distance is the best candidate loss function for learning the target distribution we chose to specify all models with this loss function when conducting our validation analysis.

6.1.1 Out-of-sample Model Predicted Portfolio Returns

To obtain the out-of-sample portfolio returns we specify a long-short and volatility scaling constraint to train both DmNN and DmLSTM, per the findings of 6.1. We feed the prior-day return into the model to avoid look-ahead bias and use 26 to calculate the out-of-sample portfolio returns. We use a transaction cost per trade, C, of one basis point 1bp, equivalent to a 0.01% of equity cost per trade, which is a low-side estimate of cost to trade, however, given all assets in our portfolio trade in liquid markets, this is not to extreme of an assumption. The green and purple empirical density's represent the out-of-sample predicted

portfolio returns when using DmNN and DmLSTM, respectively [Figure 10].

The empirical out-of-sample portfolio return distribution predicted by DmNN is observed to be approximately normal centred at zero (green density [Figure 10], suggesting that the trained model parameters where unable to find weights that map returns to the target, this suggests that DmNN may have been overfit to the training set yielding a high low bias model at the expense of high variance predictions. This conclusion is supported when comparing the in-sample optimised return density for DmNN using the Wassertein distance [Figure 8] to the green density in [Figure 10], the difference in the shape of these densities with reference to the target (blue density) is evidence of a high variance model.

Contrasting DmNN to DmLSTM we find the out-of-sample portfolio return densities to be different, with the DmLSTM density showing distribution similarities to the target [Figure 10]. Particularly the DmLSTM density is strongly positively skewed to the right-hand side, with a positive center of mass, similar to our target. Contrasting both DmNN and DmLSTM models, we can also conclude that the DmLSTMs out-of-sample portfolio return density has a much greater right tail cumulative density, suggesting the DmLSTM portfolio correctly assigns weights to long or short assets that increase or decrease in price the next day a majority of the time. It can be argued that a portfolio manager with a simple profit maximisation objective would choose the DmLSTM portfolio over the DmNN portfolio from this information alone.

6.2 Multi Portfolio Simulation

In order to gain inference on model performance we train and test our models, DmNN and DmLSTM, on different data with and without volatility scaling and either the long-short or long-only constraint. To standerdise performance, we compare our models against other benchmarks, MVO, MVO+BL and DLS [9].

6.2.1 Validation Sharpe Ratios

When allowing for volatility scaling and varying the long-only and long-short weight constraint we see that DmNN provides comparable performance too DmLSTM in the long-short case [Figure 11] and inferior performance to DmLSTM in the long-only case, when DmLSTM provides the second-best performance, strongly correlated to MVO+BL [Figure 13]. Across long-short and long-only portfolios using volatility scaling, DmNN yields the lowest average Sharpe ratio performance across all portfolios. [Figures 11, 13].

When not using volatility scaling, we implicitly stop using leveraged positions, maintaining a full investment of our wealth at each point in time. Focusing on the long-short portfolio case, we again see that DmNN provides comparable performance to DmLSTM [Figure 11], however when looking at the long-only case DmLSTM no longer outperforms DmNN and other benchmarks, rather producing similar portfolio Sharpe ratio's as DmNN and MVO [Figure 13].

6.2.2 Validation Sortino Ratios

Sortino ratio values are effectively non-linear transformations of the Sharpe ratio. This is because the downside deviation (denominator in Sortino), σ_d is effectively proportional to the full deviation, σ_P, with the proportionality constant being one over the square root of two for symmetric returns, so $\sigma_d = \frac{\sigma_P}{\sqrt{2}}$.

Given we observe our out of sample returns to be approximately symmetric [Figure 10], if we assume *iid* normal of out of sample returns, $R_P \sim N(0, \sigma_P^2)$, by Law of Large Numbers (LLN) our downside deviation estimate will tend to the root average of sum of deviations squared (the standard deviation of all returns) scaled by a factor of one over root two. Again, assuming symmetry and normality of model-predicted portfolio returns we would expect the

Sortino ratios to be only slightly larger than the Sharpe ratios. Given DmNN is observed to follow these assumptions we would expect to see this trend in our validation performance.

We in fact do observe this phenomenon, which is clear when comparing Figures 11 to 12 and 13 to 14.

For this reason, it was not surprising to find that the model vs. model and model vs. the benchmark performance were very similar to that in 6.2.1, which is again a consequence of the Sortino ratio being a scaled version of the Sharpe ratio. Although before observing the results, we hypothesised for this outcome to be unlikely, at least for DmLSTM, given our findings in 6.1 which suggested DmLSTM to produce non-normal asymmetric distributions. (which could be a consequence of us using a portfolio of 10 rather than 30 assets).

6.3 Model Risk

We refine our analysis to long-only models, given that they yield superior performance in the validation analysis. DmNN is the most risky of models, having the greatest expected loss in the worst 5% of cases compared to any model across non-volatility scaled and volatility scaled portfolios.

The effect of leverage on "return risk" is clear when comparing models that use volatility scaling against models that do not use volatility scaling. Models that use volatility scaling: DmNN, DmLSTM, and DLS, can experience single-day losses in the worst 5% of cases 300% more extreme than when using the same model without volatility scaling, highlighting the risk of volatility scaling positions.

We can also conclude that DmNN is a more risky model than DmLSTM.

6.4 Model Correlations

Focusing on models under the long-only constraint [Figure 15], we find DmNN to be the least correlated to other portfolio models, suggesting that it produces the most different set of output weights. DmLSTM close performance to MVO+BL observed in Figure 13 is again highlighted by its 0.85 correlation coefficient to MVO+BL's Sharpe Ratios. DmLSTM low Sharpe correlations with respect to other deep learning models (DmNN and DLS) imply that it produces the most unique set of portfolio returns. These findings are consistent for model Sortino Correlations in the volatility scaling and no-volatility scaling case [Figure 15] due to the proportionality phenomenon discussed in 6.2.2.

7 Limitations of Study

7.1 Arbitrary Hyperparameters

The choice of the number of layers and neurons in each layer is usually arbitrary, and we have been making this assumption so far. Layers and the number of neurons in a neural network are usually called **hyperparameters**. A hyperparameter is an exogenous value that is initialized before training starts and its values are not learned from data directly. Examples of hyperparameters include: number of epochs, dropout rate, number of layers, and number of neurons per layer.

Neural networks are universal function approximators, although poor specification of hyperparameters can severely effect model performance. A poorly specified model could have high (low) bias, meaning the model underfits (overfitting) the data, which usually comes with the expense of lower (higher) variance. What we have just defined is called the **bias-variance trade-off**. The bias-variance trade-off is identical to the decomposition of the mean square error (MSE).

$$\mathbb{E}\left[\left(R_{trg} - f_{\hat{\theta}}\right)^2\right] = \text{Bias}^2\left[f_{\hat{\theta}}\right] + \text{Var}\left(f_{\hat{\theta}}\right)$$

We can use a Bayesian optimisation scheme to find optimal hyperparameters ($\boldsymbol{\theta} = \eta, L, N$) [24], with η: learning rate, used in backpropagation, L: the number of hidden layers (total layers less 1) and N: neurons per layer.

In our case, our models seem to be overfit, DmNN more so the DmLSTM, meaning we have low bias at the expense of higher variance out-of-sample, which we observe and discuss in 6.2. Selecting suitable hyperparameters could lead to better generalisable models or contribute to further increasing model bias. Hyperparameter tuning is left for future work.

7.2 Regularisation

To penalise low model bias and mitigate high out-of-sample variance we could introduce regularisation techniques such as L1, L2 shrinkage, dropout or early stopping. All these methods penalise the model for over complexity. This would have been especially valuable for DmNN, which is a fully connected network that has a parameter for every neuron in the model. Without regularisaion models like DmNN are free to learn noise in the training data, which might explain the near-perfect in-sample performance (low bias) and poor out-of-sample performance (high variance).

7.3 Target Return Distribution Specification

Our choice of a target distribution is fixed, so we have not tested our models against benchmarks with other targets, where our models may have performed better or worse. Therefore, we cannot make general statements about the success of this approach to portfolio optimization.

Furthermore, our target distribution can be unrealistic and not attainable. For example, we could define our target return to follow a log-normal distribution, meaning our model should achieve non-negative returns. This is impossible as it would imply we have a model that never has a losing trade, always defining weights with the same sign as returns, resulting in only positive returns. Therefore, our models do theoretically have an unknown range of success, before the target distribution is never attainable, at least in the in-sample case only.

7.4 Leverage and Margin

Using volatility scaling in our models can lead to us holding significantly levered positions. Given we do not incorporate margin requirements or a maximum leverage threshold, we are not modelling out-of-sample portfolio returns for DmLSTM, DmNN and DLS as accurately as possible depsite accounting for transaction costs induced be rebalancing to new portfolio weights each new trading day. This means our model performance could be biased leading to incorrect inference on our model's success.

8 Conclusion

Traditional portfolio optimisation methods rely on *iid* joint Gaussian return distributions and fixed assumptions of investor preferences modelled using biased point estimation techniques susceptible to misspecification. To attempt to overcome the limitations of older portfolio optimisation models built on the Mean-Variance Framework (MVO) recent advancements have seen the widespread adoption of Deep Learning models. Recurrent Neural Network's (RNNs) are the most popular choice of Deep Learning models in recent portfolio optimisation literature, due to their ability to model events in the context of prior information, i.e. the ability to model dependence, which overcomes *iid* Gaussian return distribution assumptions required by standard models (MVO). Much of the literature defines these models with Sharpe ratio loss functions that are used to optimise model parameters

for "optimal" portfolio weight prediction. The Sharpe ratio embeds the preference of the greatest expected return per unit risk, which tells us nothing about the nature of the risk e.g. is positive risk more favourable than downside risk?. Informed investors with a more complex set of preferences wishing to consider a wide array of different strategies are thus stuck between traditional methods or modern methods that both suffer from a common problem, "*Lack of ability to encode investor preference in the portfolio optimisation problem*".

Our research problem is to propose a model that can account for more flexible investor preferences by leveraging the benefits of Deep Learning and probability density functions (PDFs). We know that PDFs tell us the behaviour of a given random variable, e.g. a random variable (RV) with an asymmetric distribution, many positive outliers and a "fat" right-hand tail, tells us (assuming observations follow this distribution) that we would expect observations of this process to be large and positive with high likelihood. What we have just showcased is a RVs density in the context of portfolio returns can be used to model an investor's portfolio preferences, e.g. If an asset manager wanted stable returns, he might wish to have a portfolio return distribution that is approximately normal with thin tails.

The logical next step is to ask how we can learn this distribution. Our proposed answer to this question is, to treat our preferences as some known valid distribution that we can sample from, we call this our target, and it encodes our set of preferences, we then define a deep learning model which outputs a predicted set of estimated portfolio returns. The observed estimate of portfolio returns can then be compared to the target using a distance metric that measures distribution similarity. To maximise our utility, we minimise the difference between the target and predicted portfolio return distributions. We evaluate the use of three valid distance metrics as model loss functions: The L_2^2 divergence, the unbiased squared maximum mean discrepancy (MMD_u^2) and the one-dimensional Wasserstein distance, which we use to optimise our models. To compare the suitability of different Deep Learning models we compare and contrast a simple case model, DmNN, and RNN-based model, DmLSTM in line with most modern literature.

By comparing the pre- to post-optimisation in-sample outputs of our models using different loss functions we find the Wasserstein distance is the most effective at maximising our utility (mapping returns to portfolio weight estimates which result in an estimated portfolio return density equal to our target portfolio return density). For DmNN, the Wasserstein distance can be used to find weights that map to portfolio returns which are distributed nearly identically to our target. We validate the distribution match in the DmNN case by constructing an empirical hypothesis test using a bootstrap data generating process (DGP), this test returns statistically insignificant $p - values$ that lead us to retaining the hypothesis that our densities match. We find DmLSTM to be invarient to the choice of loss function. Surprisingly DmNNs in sample performance is not replicated out-of-sample, with both models producing near normal and symmetric out-of-sample portfolio return distributions.

We find our models to perform better when using volatility scaling and enforcing the long-only weight constraint. In this setting DmLSTM is arguably the best preforming model, only slightly underperforming the best benchmark model, MVO+BL, on a Sharpe and Sortino ratio based metric. In this model specification DmLSTM is the most superior Deep Learning model, as it yields lower value at risk (VaR) and conditional value at risk (CVaR) of returns at the 5% significance level than DmNN or DLS.

DmNN is generally the worst-performing portfolio optimisation model, underperforming DmLSTM and all benchmarks based on Sharpe ratio, Sortino ratio, VaR and CVaR performance metrics, making it an undesirable portfolio optimisation model. Our results suggest that LSTM models with distribution-driven objectives and long-only constraints are a promising direction for portfolio optimisation.

Overfitting is inherent to all machine learning based models and, DmNN and DmLSTM are no exception. The lack of use of regularisation techniques results in our models potentially fitting to noise, leading to perfect in-sample performance but average out-of-sample performance. Although no point estimates are made, we implicitly assume returns are weekly stationary as a consequence of assuming model fitted parameters will map returns to weights that result in out-of-sample portfolio returns with the same distribution as the target distribution (Invokes weak stationarity as the mean and variance of these distributions must remain constant). Other model specific limitations include unrealistic definitions of the target distributions, which result in our models never finding weights that map to returns that approximate the target density, at least in the in-sample case.

We have presented a novel approach to the portfolio optimisation problem that overcomes the problem of simple preferences. We have used Deep Learning models optimised using distance metric loss functions in an attempt to map returns to weights that result in estimated portfolio returns that match our target preferences. We enforced different constraints in our problem, from long-short, long-only, to leverage via volatility scaling. Our findings suggest that the DmLSTM model in the long-only and leverage setting was the best performing Deep-Learning model, out-competing "state of the art" models in recent literature such as DLS [10]. However, there is much scope for improvement in our models and research framework that we will leave for future work. These include the addition of regularisation to improve out-of-sample density matching, modification to the calculation of out-of-sample returns to account for trading on margin, and an analysis of model performance across a representative set of realistic target distributions.

9 References

[1] - Beasley, J.E., 2013. Portfolio optimisation: models and solution approaches. In Theory Driven by Influential Applications (pp. 201-221). Informs.

[2] - Harry Markowitz. Portfolio selection. The journal of finance, 7(1):77–91, 1952.

[3] - Chang, Y.J., Wang, W.T., Wang, Y.Y., Liu, C.Y., Chen, K.C. and Chang, C.R., 2025. Quantum Stochastic Walks for Portfolio Optimization: Theory and Implementation on Financial Networks. arXiv preprint arXiv:2507.03963.

[4] - Chang, T.J., Meade, N., Beasley, J.E. and Sharaiha, Y.M., 2000. Heuristics for cardinality constrained portfolio optimisation. Computers & Operations Research, 27(13), pp.1271-1302.

[5] - Sharpe, W.F., 1964. Capital asset prices: A theory of market equilibrium under conditions of risk. The journal of finance, 19(3), pp.425-442.

[6] - Black, F. and Litterman, R., 1990. Asset allocation: combining investor views with market equilibrium. Goldman Sachs Fixed Income Research, 115(1), pp.7-18.

[7] - Idzorek, T., 2007. A step-by-step guide to the Black-Litterman model: Incorporating user-specified confidence levels. In Forecasting expected returns in the financial markets (pp. 17-38). Academic Press.

[8] - Robert Andrew Matrin Revision 48659587 2019, *PyPortfolioOpt Black-Litterman Allocation*, accessed 3/08/2025, <https://pyportfolioopt.readthedocs.io/en/latest/BlackLitterman.html#id6>

[9] - Novykov, V., Bilson, C., Gepp, A., Harris, G. and Vanstone, B.J., 2025. Deep learning applications in investment portfolio management: a systematic literature review. Journal of Accounting Literature, 47(2), pp.245-276.

[10] - Zhang, Z., Zohren, S. and Roberts, S., 2020. Deep learning for portfolio optimization. arXiv preprint arXiv:2005.13665.

[11] - Karzanov, D., Garzón, R., Terekhov, M., Gulcehre, C., Raffinot, T. and Detyniecki, M., 2025. Regret-Optimized Portfolio Enhancement through Deep Reinforcement Learning and Future Looking Rewards. arXiv preprint arXiv:2502.02619.

[12] - Kim, J., 2025. Semi-Decision-Focused Learning with Deep Ensembles: A Practical Framework for Robust Portfolio Optimization. arXiv preprint arXiv:2503.13544.

[13] - Staudemeyer, R.C. and Morris, E.R., 2019. Understanding LSTM–a tutorial into long short-term memory recurrent neural networks. arXiv preprint arXiv:1909.09586.

[14] - 3Blue1Brown 11/03/2017, *Backpropagation calculus | Deep Learning Chapter 4*, accessed 25/08/2025, <https://www.youtube.com/watch?v=tIeHLnjs5U8>

[15] - Colah's blog 27/08/2015, *Understanding LSTM Networks*, accessed 22/08/2025, <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

[16] - Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural networks, 4(2), pp.251-257.

[17] - Thorir mar Ingolfsson 10/11/2021, *Insights into LSTM architecture*, accessed 23/08/2025, <https://thorirmar.com/post/insight_into_lstm/>

[18] - Deisenroth, M.P., Faisal, A.A. and Ong, C.S., 2020. Mathematics for machine learning. Cambridge University Press.

[19] - Krishnamurthy, A., Kandasamy, K., Poczos, B. and Wasserman, L., 2015, February. On Estimating L_2^2 Divergence. In Artificial Intelligence and Statistics (pp. 498-506). PMLR.

[20] - Gretton, A., Borgwardt, K.M., Rasch, M.J., Schölkopf, B. and Smola, A., 2012. A kernel two-sample test. The journal of machine learning research, 13(1), pp.723-773.

[21] - Panaretos, V.M. and Zemel, Y., 2019. Statistical aspects of Wasserstein distances. Annual review of statistics and its application, 6(1), pp.405-431.

[22] - ABDULLAH 2022, *Most Watched Stocks of Past Decade(2013-2023)*, accessed 15/08/2025, <https://www.kaggle.com/datasets/kane6543/most-watched-stocks-of-past-decade20132023>

[23] - Belsley, D.A. and Kontoghiorghes, E.J. eds., 2009. Handbook of computational econometrics (p. 172). New York: Wiley.

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

[25] - Amjad Saidam 11/09/2025, STAT0034-Research-Project- <https://github.com/AmjadSaidam/STAT0034-Research-Project-.git>

10 Appendix

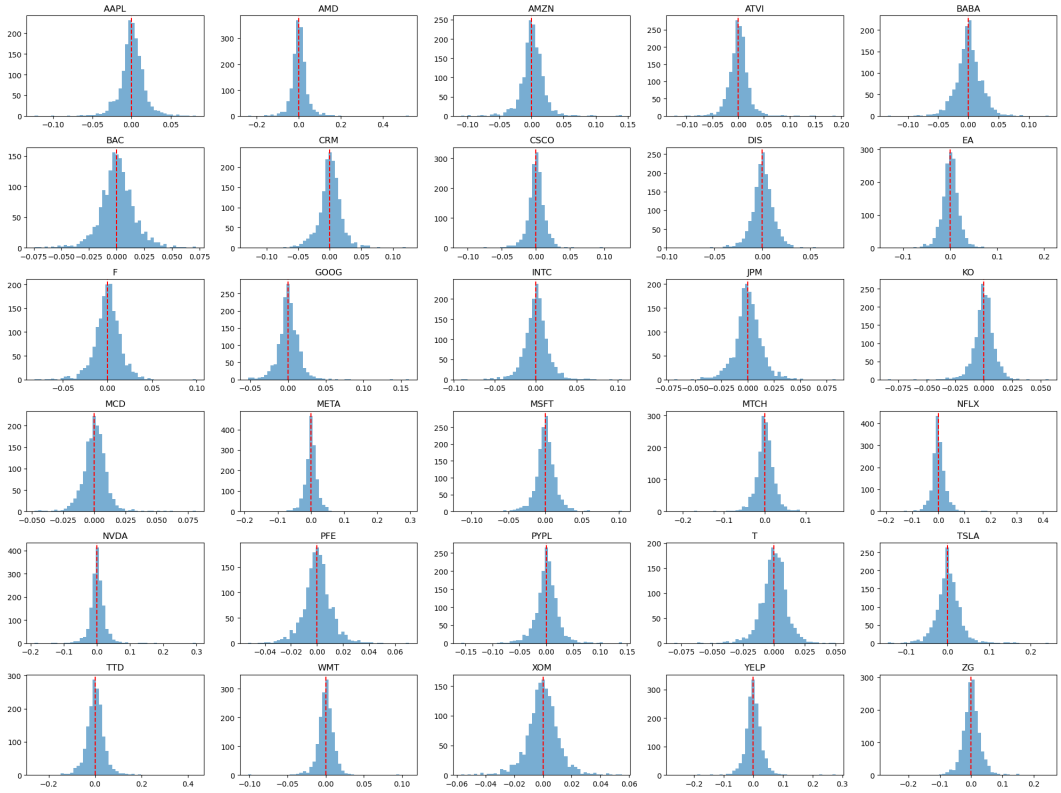

Figure 16: A discrete density plot of all our asset returns. All returns are calculated using the simple asset returns discussed in [1]

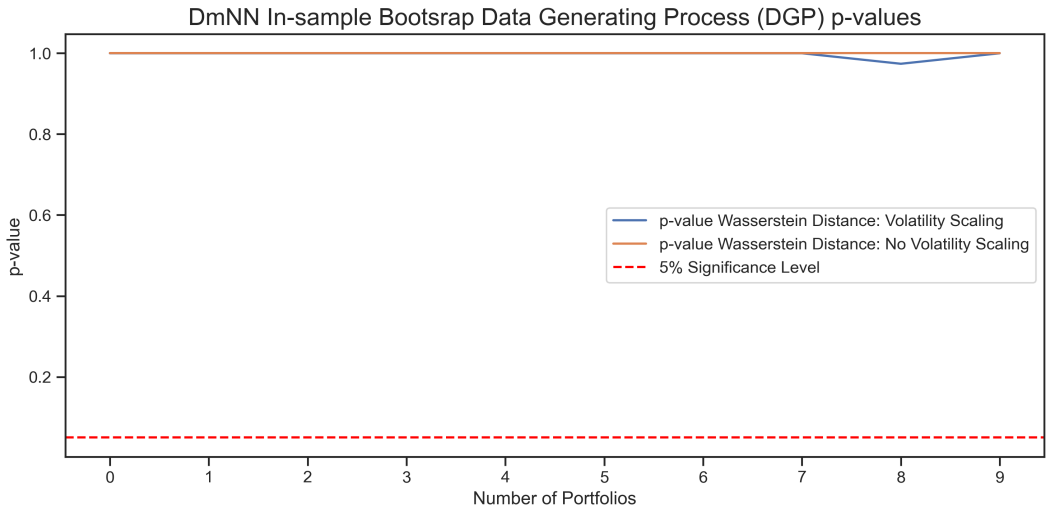

Figure 17: $p-values$ of hypothesis test for return distribution similarity between $\hat{R}$ and R^* using a bootstrap data generating process (DGP). Where $\hat{R}$ is produced from a 1000 epoch trained DmNN model, using volatility scaling and the long-short constraint.

CVaR of Models and Benchmarks: Long-only, Volatility Scaling				
CVaR DmNN	**CVaR DmLSTM**	**CVaR MVO**	**CVaR MVO+BL**	**CVaR DLS**
-8.70%	-7.32%	-3.21%	-2.94%	-8.33%
-9.44%	-7.21%	-3.48%	-3.15%	-7.54%
-10.41%	-6.33%	-3.10%	-3.08%	-7.95%
-9.83%	-6.53%	-3.10%	-3.06%	-8.18%
-9.74%	-7.14%	-3.10%	-3.26%	-7.41%
-10.15%	-7.89%	-3.54%	-3.45%	-8.60%
-11.14%	-6.57%	-2.96%	-3.11%	-8.09%
-9.15%	-6.16%	-3.00%	-3.05%	-7.52%
-11.16%	-6.38%	-2.95%	-2.90%	-7.96%
-9.89%	-6.88%	-3.10%	-3.34%	-7.46%

Table 4: Conditional Value at Risk (CVaR) of predicted portfolio returns on out-of-sample data for long only DmNN and DmLSTM models with volatility scaling vs. Benchmarks, MVO, MVO+BL and DLS.

CVaR of Models and Benchmarks: Long-only, Volatility Scaling				
CVaR DmNN	**CVaR DmLSTM**	**CVaR MVO**	**CVaR MVO+BL**	**CVaR DLS**
-7.28%	-2.53%	-3.21%	-2.94%	-3.10%
-7.17%	-2.77%	-3.48%	-3.15%	-3.54%
-7.16%	-2.52%	-3.10%	-3.08%	-3.18%
-6.17%	-2.52%	-3.10%	-3.06%	-3.54%
-7.21%	-2.63%	-3.10%	-3.26%	-3.45%
-7.20%	-2.90%	-3.54%	-3.45%	-3.50%
-7.18%	-2.63%	-2.96%	-3.11%	-3.25%
-7.12%	-2.56%	-3.00%	-3.05%	-3.38%
-4.71%	-2.45%	-2.95%	-2.90%	-2.80%
-7.15%	-2.51%	-3.10%	-3.34%	-3.55%

Table 5: Conditional Value at Risk (VaR) of predicted portfolio returns on out-of-sample data for long only DmNN and DmLSTM models with no volatility scaling vs. Benchmarks, MVO, MVO+BL and DLS.