

Deep Graph Convolutional Reinforcement Learning for Financial Portfolio Management - DeepPocket

Farzan Soleymani, Eric Paquet*

National Research Council, 1200 Montreal Road, Ottawa, ON K1K 2E1, Canada

arXiv:2105.08664v1 [q-fin.CP] 6 May 2021

Abstract

Portfolio management aims at maximizing the return on investment while minimizing risk by continuously reallocating the assets forming the portfolio. These assets are not independent but correlated during a short time period. A graph convolutional reinforcement learning framework called DeepPocket is proposed whose objective is to exploit the time-varying interrelations between financial instruments. These interrelations are represented by a graph whose nodes correspond to the financial instruments while the edges correspond to a pair-wise correlation function in between assets. DeepPocket consists of a restricted, stacked autoencoder for feature extraction, a convolutional network to collect underlying local information shared among financial instruments and an actor-critic reinforcement learning agent. The actor-critic structure contains two convolutional networks in which the actor learns and enforces an investment policy which is, in turn, evaluated by the critic in order to determine the best course of action by constantly reallocating the various portfolio assets to optimize the expected return on investment. The agent is initially trained offline with online stochastic batching on historical data. As new data become available, it is trained online with a passive concept drift approach to handle unexpected changes in their distributions. DeepPocket is evaluated against five real-life datasets over three distinct investment periods, including during the Covid-19 crisis, and clearly outperformed market indexes.

Keywords: Portfolio Management, Deep Reinforcement learning, Restricted Stacked Autoencoder, Online Learning, Actor-Critic, Graph Convolutional Network

1. Introduction

The process of selecting and managing a group of financial instruments such as stocks, bonds, and securities, is called portfolio management (Markowitz, 1978). Portfolio management aims at maximizing the return on investment while minimizing the risk. Stock market movements vary with time and reflect social and political trends. As a result, the financial market follows a complex trajectory that is characterized by its volatility as well as by the correlations between financial instruments. One must have a broad knowledge of these movements and erratic behaviors in order to somehow predict market evolution. Various strategies for portfolio management have been proposed in the literature such as (Pouya et al., 2016) which employs a multiobjective weed optimization method, (Yue & Wang, 2017) which relies on fuzzy multi-objective, high-order moment portfolio selection, and (Omidi et al., 2017) which dynamically solves portfolio selection with uncertainty chance constraints. Constrained portfolio selection has been discussed in (García et al., 2019a,b). In this paper, it is proposed to learn the portfolio allocation directly from the data with a deep neural network: a free-form, data-driven approach.

Investing in multiple indexes and stocks could be overwhelming as one must assess market evolution while continuously reallocating assets in the most profitable way. Market movements, price fluctuations, and sudden turmoil are known to be interrelated (Park & Shin, 2013). Yet, it is extremely difficult to model the duration and magnitude of these fluctuations (Sarwar & Khan, 2019; Park & Shin, 2013). For instance, the financial market is influenced by exogenous factors related to the globalized economy, such as the merging of 21st Century Fox and Disney, or Peugeot Fiat-Chrysler. Understanding the dependencies and interrelationships among assets helps to understand market evolution while identifying the optimal portfolio composition and risk management strategy (Drożdż et al., 2001; Elton et al., 2009). Deep learning techniques are well suited for extracting complex patterns

*Corresponding author

Email addresses: Farzan.Soleymani@nrc-cnrc.gc.ca (Farzan Soleymani), Eric.Paquet@nrc-cnrc.gc.ca (Eric Paquet)

from large datasets (Bengio et al., 2015). In addition, datasets are mostly Euclidean in the sense that the space associated with the data has a Euclidean geometry.

Images are a good example of such a geometry as the pixels form a regular grid which spans a bidimensional Euclidean space. An important benefit of Euclidean space is that it permits the concept of convolution. This is paramount for convolutional networks as their architecture is based on this very concept. Convolutional neural networks (CNNs) have been widely employed for pattern recognition, localization, and parameter reduction on Euclidean datasets, just to mention a few. If the space is not Euclidean, convolution networks cannot be employed in their standard form (Defferrard et al., 2016). Moreover, most deep learning algorithms assume that the instances forming the dataset are statistically independent (Gama et al., 2013; Barabási et al., 2016; Wu et al., 2019); but this assumption does not apply to datasets whose instances are interrelated. Still, non-Euclidean data are far from rare. One may think, for instance, of scientific publications and citations (Defferrard et al., 2016; Veličković et al., 2017) in which the various instances are linked to each other, thereby forming a non-Euclidean interconnected network. The same may be said about macromolecular structures, whose importance in medicine cannot be overstressed (Gilmer et al., 2017).

Financial data, such as financial instruments, are no exception as they also define a non-Euclidean space (Wu et al., 2020). Indeed, the reason lies within the interrelation between financial instruments (Lucey & Muckley, 2011). These are correlated (Anthony, 1988) during short periods of time, especially in the presence of exogenous extreme events (Arfaoui & Rejeb, 2017). Their structure may be represented by a graph in which the nodes correspond to the financial instruments, while the edges or links correspond to some pair-wise correlation function. Because the geometry is non-Euclidean, one cannot easily transpose the Euclidean definitions of locality, translation, and compositionality which underpin convolution and, ultimately, are needed to employ convolutional neural networks (Angles & Gutierrez, 2008; Bronstein et al., 2017). In Euclidean geometry, convolution may be performed directly or with the help of the convolution theorem (Hammond et al., 2011) by employing the Fourier transform. The latter is well defined in Euclidean space.

In order to apply the convolution theorem in finance, one must define a Fourier transform directly on the non-Euclidean graph. Such a graph Fourier transform was introduced recently by (Shuman et al., 2013): it allows convolutions on the graph to be evaluated directly with the help of the convolution theorem and the graph Fourier transform. Once defined, the transform makes it possible to apply a convolutional network directly to non-Euclidean graphs, which are henceforth known as graph convolutional networks (GCN) (Shuman et al., 2013; Henaff et al., 2015). As for their Euclidean counterparts, they consist of a bank of kernels (or filters) which are convolved with their input data. Euclidean filters are local in the sense that their extension (dimensionality) is much smaller than that of the signals with which they are convolved. In the non-Euclidean case, because of the Fourier transform, localization is lost, generally speaking. This problem may be overcome if the filters are defined with truncated Chebyshev polynomials, which are characterized by their compact support and low computational complexity (recurrence relation) (Defferrard et al., 2016).

The portfolio management problem is inherently nonlinear, stochastic, and time-dependent. Therefore, it may be portrayed as a decision-making process that yields a sequence of actions. These actions are defined as the amount of funds to be allocated to each asset forming the portfolio to increase the expected return on investment. This is achieved by training an agent which selects a set of actions from all possible actions in the action space, aiming to increase the portfolio return over a certain investment period. This objective may be formulated as a stochastic optimization problem such that the solution yields an optimal chain of actions which aims at maximizing the expected return on investment.

These types of problem can be solved using reinforcement learning (RL) techniques where an agent takes action (\mathbf{a}_t) from an action space (\mathcal{A}) based on the state (\mathbf{s}_t) of the environment which belongs to a state-space (\mathcal{S}). Each action can be interpreted as a kind of interaction with the environment, which is associated with a scalar reward r_t . RL determines an optimal path in the action space, which aims at increasing the reward. The path is characterized by a chain of actions that are derived from a policy function $\pi(\mathbf{a}_t|\mathbf{s}_t)$. Therefore, the agent learns a policy based on past experience, which is evaluated by a value function according to some environment dynamics (Andrew, 1999). Since the portfolio management problem is time-dependent, learning must be conducted online.

There are various RL techniques that provide optimal policy and value functions, e.g. Q-learning (Watkins & Dayan, 1992; Bu et al., 2008), SARSA (Zhao et al., 2016), deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015) and actor-critic (Konda & Tsitsiklis, 2000) just to mention a few. Q-learning is an off-policy algorithm meaning that the policy that generates the actions may be unrelated to the improved policy. On the other hand, SARSA is an on-policy approach that aims to improve the policy that dictates the actions (Andrew, 1999). DDPG is a model-free, off-policy approach that learns a deterministic policy in a continuous space (Lillicrap et al., 2015). The advantage of such a policy is its ability to explore all possible actions while learning a deterministic policy

(Andrew, 1999). Nonetheless, this exploration is time-consuming as it requires a large number of training epochs. In the DeepBreath framework, a SARSA algorithm was employed in order to learn the optimal policy (Soleymani & Paquet, 2020). However, SARSA may lead to abrupt changes in the policy which are not necessarily suitable for portfolio optimization (Celikyurt & Özekici, 2007) as a result of an erratic change in the policy if the latter performs too poorly (Andrew, 1999).

The actor-critic algorithm consists of two components: one being associated with the policy while the other is associated with the value function. The network responsible for the implementation of the policy is known as the actor. The critic is a learnable network that evaluates the performance index (value function) for each new each state based on the actions previously taken by the actor. Therefore, the actor-critic approach may be implemented with two online, trainable neural networks that work concurrently in tandem. In this work, the actor-critic algorithm is exploited wherein the actor is trained based on experience, corresponding to probabilistic mapping from state actions according to a learnable policy. At the same time, the critic estimates the expected future return through a state-value function while constantly improving the policy (Bhatnagar et al., 2009). Both the actor and the critic are implemented with specially designed convolutional networks where the output of the actor determines the portfolio allocation while the return on investment is approximated by the critic. One of the most important innovations in this work is the introduction of a **graph convolutional network** for the actor and the critic in order to take advantage of the correlation (as well as the non-Euclidean space) existing in between the financial instruments.

A graph convolutional reinforcement learning framework, DeepPocket is introduced, consisting of a restricted, stacked autoencoder (RSAE) for feature extraction and dimensionality reduction, a graph convolutional network (GCN) to acquire interrelations among financial instruments, as well as a convolutional network for each of the actor and the critic. Feature extraction generates a low-dimensional and information-rich description which is more suited to machine learning, besides reducing the computational complexity. The GCN captures the correlations existing between financial instruments. The investment policy is enforced, and the return on investment is estimated using the actor-critic algorithm. The framework is initially trained offline; then, the weights of the neural networks are updated online as new data becomes available. The offline training step is conducted using historical data, while online training is based on passive concept-drift detection (Gama et al., 2014).

The paper is organized as follows. The mathematical model of portfolio management is discussed in Section 2. The architecture of the proposed framework is presented in Section 3, which includes feature normalization in subsection 3.1, the restricted stacked autoencoder in subsection 3.2, and the graph convolutional networks in subsection 4. The reinforcement learning framework is introduced in Section 5. This is followed by a description of our offline and online learning approaches in Section 6. Our experimental results are presented in Section 7. Finally, Section 8 concludes the paper.

2. Mathematical Model

A portfolio containing m assets may be managed directly by an individual or by financial professionals through an institution like an investment bank. The investors must select and allocate funds to a group of assets according to an investment strategy, subject to some risk aversion. Our portfolio management model is inspired by an approach introduced by (Ormos & Urbán, 2013) which introduces an experimental approximation of the log-optimal investment strategy which ensures a near optimal growth rate of investments along with a survivorship bias-free setup. In this model, a trading period corresponds to one business day. The closing price of all assets in the portfolio corresponds to the vector \mathbf{V}_t . The vector also contains the amount of available cash. The normalized price vector is defined as

$$\mathbf{Y}_t := \mathbf{V}_t \oslash \mathbf{V}_{t-1} = \left[1, \frac{\mathbf{V}_{1,t}}{\mathbf{V}_{1,t-1}}, \frac{\mathbf{V}_{2,t}}{\mathbf{V}_{2,t-1}}, \dots, \frac{\mathbf{V}_{m,t}}{\mathbf{V}_{m,t-1}} \right]^T. \quad (1)$$

where (\oslash) denotes element-wise division. Initially, there is no assumption regarding the weight distribution: as a result, the portfolio only consists of cash which means that the weight associated with the cash is one (1), while the other weights are zero (0):

$$\mathbf{w}_0 = [1, 0, \dots, 0]^T. \quad (2)$$

A transaction factor, which shrinks the portfolio value, is associated with each transaction, such as buying or selling. At the end of a business day, as illustrated in Fig. 1, the portfolio weights vector is equal to:

$$\mathbf{w}'_t = \frac{\mathbf{Y}_t \odot \mathbf{w}_{t-1}}{\mathbf{Y}_t \cdot \mathbf{w}_{t-1}}, \quad (3)$$

At the beginning of the period (t), the portfolio weight vector is \mathbf{w}_{t-1} . The agent enforces the investment policy in order to reallocate the assets, such as to increase the expected return on investment, resulting, at the end of the trading day, in a new weight vector \mathbf{w}'_t . A commission fee $\mu \in (0, 1]$ is applied to the transactions, resulting in the final weight vector \mathbf{w}_t .

$$P_t = \mu_t P'_t. \quad (4)$$

The portfolio value at the end of period (t) is obtained by

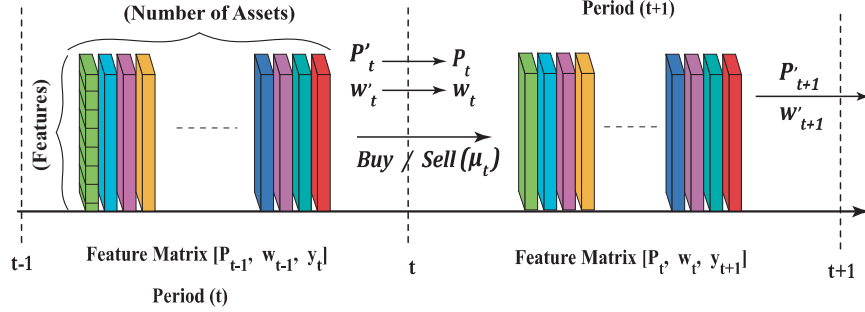


Figure 1: Weight reallocation process. The portfolio weight vector \mathbf{w}'_{t-1} and the portfolio value P'_{t-1} at the beginning of period (t) evolve to (t) and P'_t respectively. At that moment, assets are sold and bought in order to increase the expected return on investment. These operations involve commission fees which shrink the portfolio value to P_t which result in new weights \mathbf{w}_t .

$$P_t = \mu_t P_{t-1} \cdot \mathbf{Y}_t \cdot \mathbf{w}_{t-1}. \quad (5)$$

while the return rate and logarithmic return rate for period (t) are given by

$$\rho_t := \frac{P_t}{P_{t-1}} - 1 = \frac{\mu_t \cdot P'_t}{P_{t-1}} - 1 = \mu_t \mathbf{Y}_t \cdot \mathbf{w}_{t-1} - 1, \quad (6a)$$

$$R_t := \ln \frac{P_t}{P_{t-1}} = \ln(\mu_t \mathbf{Y}_t \mathbf{w}_{t-1}) - 1. \quad (6b)$$

Accordingly, the final portfolio value at time horizon t_f is determined by

$$P_f = P_0 \cdot \exp \left(\sum_{t=1}^{t_f+1} R_t \right) = P_0 \prod_{t=1}^{t_f+1} \mu_t \mathbf{Y}_t \mathbf{w}_{t-1}. \quad (7)$$

Buying and selling assets result in spending and gaining cash (Soleymani & Paquet, 2020), which implies that:

$$C_g = (1 - c_s) P'_t \sum_{i=1}^m \cdot \text{ReLU}(\mathbf{w}'_{t,i} - \mu_t \mathbf{w}_{t,i}) \quad (8)$$

$$C_s = (1 - c_b) \left[\mathbf{w}'_{t,0} + (1 - c_s) \sum_{i=1}^m \text{ReLU}(\mathbf{w}'_{t,i} - \mu_t \mathbf{w}_{t,i}) - \mu_t \mathbf{w}_{t,0} \right] = \sum_{i=1}^m \text{ReLU}(\mu_t \mathbf{w}_{t,i} - \mathbf{w}'_{t,i}) \quad (9)$$

where the selling commission rate belongs to $0 < c_s < 1$, the buying commission rate lies within $0 < c_b < 1$ and $\text{ReLU}(x) = \max(0, x)$ is a rectified linear unit. As a result, the available cash $P'_t \mathbf{w}'_{t,0}$ becomes $\mu_t P'_t \mathbf{w}'_{t,0}$. Therefore, the transaction factor μ_t is obtained by solving an implicit equation:

$$\mu_t = \frac{1}{1 - c_b \mathbf{w}'_{t,0}} \left[1 - c_b \mathbf{w}'_{t,0} - (c_s + c_b - c_s c_b) \sum_{i=1}^m \text{ReLU}(\mathbf{w}'_{t,i} - \mu_t \mathbf{w}_{t,i}) \right] \quad (10a)$$

$$\mu_t = \mu_t(\mathbf{w}_{t-1}, \mathbf{w}_t, \mathbf{Y}_t) \quad (10b)$$

Eq. 10 cannot be solved analytically. A solution may, however, be achieved iteratively, convergence being guaranteed by a theorem demonstrated in (Jiang et al., 2017). Our mathematical model relies on two assumptions:

- Trading is possible at any time during business hours.
- The volume of financial instruments traded by the agent is relatively small compared to the total number of assets traded during a day period.

As long as the volume of financial instruments traded by the agent is small compared to the overall traded volume, the latter assumptions remain valid. The architecture of the proposed framework is explained in the following section.

3. System Architecture

A profitable trading system must be able to forecast price movements on the stock market. These price movements may be characterized by various financial indicators such as price trend (Nti et al., 2019), the momentum indicator, and moving average, to mention a few (see Table 1).

The price trend indicates the direction toward which the market evolves (Qiu & Song, 2016). The strength of a trend, as well as the likeliness of its reversal, are measured by the momentum indicator, this being related to the rate at which prices change (Cervelló-Royo & Guijarro, 2020). The moving average indicator measures the average price of a given financial instrument over a certain period of time. Seven financial indicators were selected in order to measure price movement and predict future trends. These indicators include the opening and closing price of the market and the lowest, and the highest prices reached during the trading period, along with financial indicators such as the Hull moving average, average true range, the dynamic momentum index, the commodity selection index, among others (Murphy, 1999). These indicators are described in Table 1.

The proposed portfolio management framework consists of multiple components, namely the normalization module, feature selection with a restricted, stacked autoencoder, and a graph convolutional neural network which extracts information-rich features based on the correlations between financial instruments. The policy is learned with an actor-critic algorithm: the latter comprising two convolutional neural networks as illustrated in Fig. 2. These neural networks are fully described in the next section.

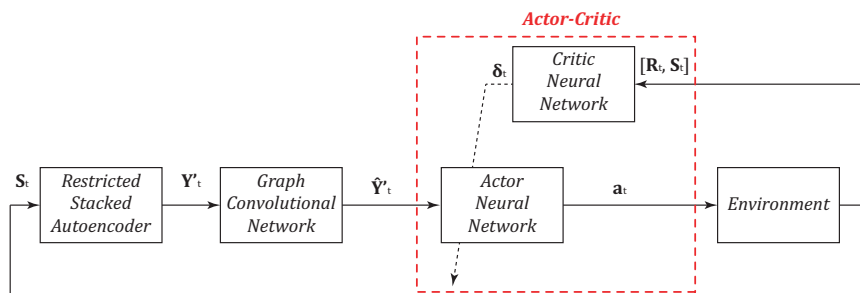


Figure 2: Global architecture of DeepPocket.

3.1. Feature Normalization

Each asset is characterized by a feature vector containing twelve (12) features including opening, closing, low and high prices in addition to financial indicators such as average true range that evaluate the market volatility over a certain period (Soleymani & Paquet, 2020) as illustrated in Table 1.

Table 1: Financial Indicators Employed as Features.

Financial Indicators	Definition
<i>Average True Range</i>	The average true range is a technical indicator that assess the volatility of an asset in the market through a certain period.
<i>Commodity Channel Index</i>	The commodity channel index is a momentum-based oscillator used to help determine cyclical trend of an asset price as well as strength and direction of that trend.
<i>Commodity Selection Index</i>	The commodity selection index is a momentum indicator that evaluates the eligibility of an asset for short term investment.
<i>Demand Index</i>	The demand index is an indicator that uses price and volume to assess buying and selling pressure affecting a security.
<i>Dynamic Momentum Index</i>	The dynamic momentum index determines if an asset is an asset is overbought or oversold.
<i>Exponential Moving Average</i>	An exponential moving average is a technical indicator that follow the price of an asset while prioritizing on recent data points.
<i>Hull Moving Average</i>	The hull moving average is a more responsive alternative to moving average indicator that focus on the current price activity whilst maintaining curve smoothness.
<i>Momentum</i>	The momentum in a technical indicator that determines the speed at which the price of an asset is changing.

Since the effective range and extrema of the features are unknown a priori, the min-max approach cannot be applied (Hussain et al., 2008; Bhanja & Das, 2018). Instead, the prices are normalized with respect to their values at opening:

$$\begin{aligned}
\mathbf{V}_t^{(Lo)} &= \left[\frac{Lo(t-n-1)}{Op(t-n-1)}, \dots, \frac{Lo(t-1)}{Op(t-1)} \right]^T, \\
\mathbf{V}_t^{(Cl)} &= \left[\frac{Cl(t-n-1)}{Op(t-n-1)}, \dots, \frac{Cl(t-1)}{Op(t-1)} \right]^T, \\
\mathbf{V}_t^{(Hi)} &= \left[\frac{Hi(t-n-1)}{Op(t-n-1)}, \dots, \frac{Hi(t-1)}{Op(t-1)} \right]^T.
\end{aligned} \tag{11}$$

while the financial indicators are normalized with respect to their closing value on the previous day:

$$\mathbf{V}_t^{(FI)} = \left[\frac{FI(t-n)}{FI(t-n-1)}, \dots, \frac{FI(t)}{FI(t-1)} \right]^T. \tag{12}$$

Feature extraction is performed on these normalized vectors with a restricted autoencoder.

3.2. Restricted Stacked Autoencoder

In order to implement machine learning algorithms efficiently, it is important to reduce the computational cost and time required for training (Meng et al., 2017). For this purpose, feature extraction is performed to obtain highly informative abstract features with low dimensionality. The input feature vector contains eleven (11) features, namely: normalized low, high, and closing prices along with the eight normalized financial indicators, all taken at the same time t . These features are partially correlated and, as a result, carry redundant information. Feature extraction aims to remove such a redundancy while making the indicators more informative. The restricted, stacked autoencoder transforms recurrent events into low-dimension abstractions (Langr & Bok, 2019).

The restricted autoencoder consists of two parts, namely the encoder and the decoder. The encoder maps the input feature to a lower dimension via multiple stacked layers, collectively called the latent layer, while the decoder reconstructs low-dimensional features at the output. The decoder is designed to reconstruct only a subset of the original features; in our case, the low, close, and high price. In order to partially reconstruct the normalized input vector, each layer of the decoder has three (3) neurons which restrict the reconstruction to the normalized low, high, and closing prices. The network is under-complete, which means the latent layer has fewer nodes than the input layer (Bengio et al., 2015). The latent layer creates a tensor of abstract features as $([V_1, V_2, V_3])$. The structure of our RSAE is illustrated in Fig. 3

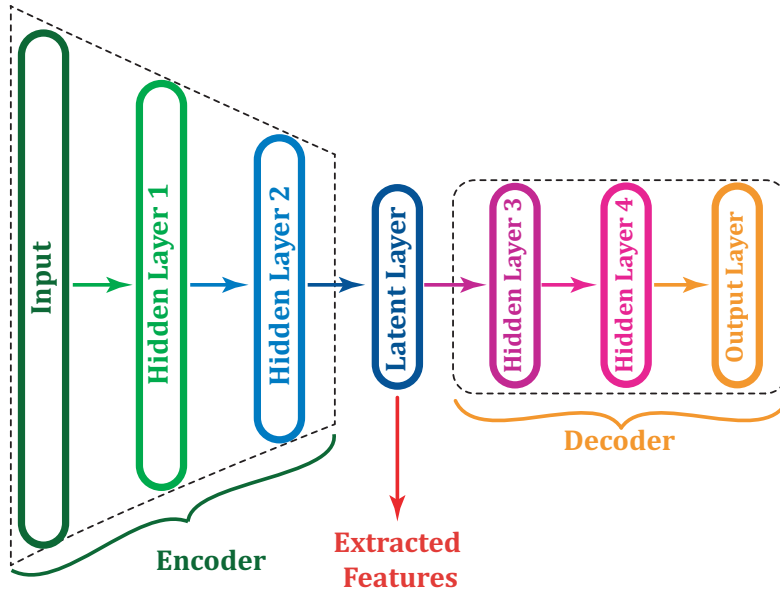


Figure 3: Restricted stacked autoencoder.

In the next section, the correlation between financial assets is extracted using graph convolutional networks.

4. Spectral Graph Convolutional Networks

Most of the data used in deep learning have a regular grid structure (Bronstein et al., 2017). One may mention, for instance, the regular pixel structure of an image or times series at fixed time intervals. These data may easily be encoded for neural network input as vectors, matrices, or tensors. These structures are readily represented in Euclidean space, exhibit properties such as local-connectivity, shift-invariance, and compositionality, to mention just a few (Stone et al., 2017). As a result, the convolution operation is properly defined, thus providing the basis for convolutional neural networks (Henaff et al., 2015).

In contrast, many entities, such as financial instruments, cannot be represented on a regular grid due to the complex nature of their interrelations. Indeed, financial instruments are correlated among themselves, and their structures may be better represented by a graph in which the nodes correspond to the financial instruments while the edges or links correspond to some pair-wise correlation function. It is worth noting that a single financial instrument, being a time series, embeds naturally in Euclidean space, but a plurality of financial instruments, because of their mutual correlations, resists Euclidean representation and instead is represented as a graph (Wu et al., 2019; Zhang et al., 2019). Unfortunately, standard convolution cannot be directly extended to non-Euclidean geometries (Shuman et al., 2013) therefore impeding its applicability to CNN. Nonetheless, as set out by the convolution theorem (Shuman et al., 2013), the convolution may be evaluated with the help of the Fourier transform: firstly, the Fourier transforms of both the input and the filter are evaluated; then, both transformations are multiplied element-by-element (the Hadamard product) and, finally, the inverse Fourier transform of the Hadamard product is taken. The convolution theorem remains valid under non-Euclidean geometry if the Fourier transform is properly defined (Shuman et al., 2013; Henaff et al., 2015), thus allowing the application of CNN to non-Euclidean geometries.

The next section provides a mathematical formulation of spectral graph convolution.

4.1. Graph Fourier transform

Let us consider an undirected weighted graph $G = \{V, E, \mathbf{W}\}$ representing, for instance, m financial instruments. This graph consists of ($|V| = m$) vertices or nodes and ($|E| = n$) edges or links. The edges represent the

interrelations between the nodes while the weights are a pair-wise correlation function between nodes. This graph may be characterized by a weight adjacency matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$:

$$\mathbf{W} = [w_{ij}] \quad (13)$$

where w_{ij} is the weight between node i and j (Shuman et al., 2016). Such a graph is dynamic in the sense that the correlations vary at each time interval. The correlation is evaluated over a period consisting of n time intervals. This period should be relatively short as the correlation tends to disappear rapidly due to randomization (Fama, 1965; Rocchi et al., 2017). For the correlation, a value of one indicates a perfect correlation while a value of zero indicates its absence. It is desirable that correlated nodes be topologically close to each other while uncorrelated nodes are more aloof. In order to reflect this behavior, the weights are defined as

$$w_{ij} = 1 - \text{corr}(V_i, V_j)_{[t-n+1, t]} \quad (14)$$

where the correlation is evaluated on the interval $[t - n + 1, t]$. A graph may be represented by a Laplacian. The standard graph Laplacian is defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \quad (15)$$

where $\mathbf{D} \in \mathbb{R}^{n \times n}$ is diagonal degree matrix that indicates the degree of connectivity of each node $d_{i,j} = \sum w_{ij}$ and \mathbf{W} is the weight matrix (Chung & Graham, 1997). This Laplacian is a discrete representation of the Laplacian operator which is encountered, for instance, in the heat and Schrödinger equations. As such, the Laplacian completely characterizes a Gaussian random walk on the graph (Felmer et al., 2012). The Laplacian may also be defined as a symmetrical matrix (Kipf & Welling, 2016):

$$\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-\frac{1}{2}} \quad (16)$$

Symmetry is highly desirable as the eigenvalues ($0 = \lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_{n-1}$) associated with a symmetric matrix will then be real while the eigenvectors are mutually orthogonal, which greatly reduces the computational complexity. The eigendecomposition of a symmetrical Laplacian is given by

$$\mathbf{L}_{sym} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^T \quad (17)$$

where $\mathbf{\Phi} = [\phi_0, \dots, \phi_{n-1}]$ are the column-wise orthogonal eigenvectors:

$$\mathbf{\Phi}^T \mathbf{\Phi} = \mathbf{I} \quad (18)$$

The eigenvalues are represented as a diagonal matrix $\mathbf{\Lambda} = \text{diag}(\lambda_0, \lambda_1, \dots, \lambda_{\max}) \in \mathbb{R}^{n \times n}$. The eigenvectors corresponding to the smallest eigenvalues are the smoothest in the sense that $|\phi_l(i) - \phi_l(j)|$ is small for two neighboring nodes i and j (Shuman et al., 2011). As aforementioned, graph convolution may be defined using the Fourier transform and the convolution theorem (Shuman et al., 2013). The graph Fourier transform of a signal \mathbf{x} on the graph is defined as

$$\mathcal{F}(\mathbf{x}) = \mathbf{\Phi}^T \mathbf{x} = \tilde{\mathbf{x}} \quad (19)$$

where $\mathbf{\Phi}^T$ indicates the transpose of $\mathbf{\Phi}$; whereas the inverse graph Fourier transform is obtained by

$$\mathcal{F}^{-1}(\tilde{\mathbf{x}}) = \mathbf{\Phi}\tilde{\mathbf{x}} = \mathbf{x} \quad (20)$$

where \mathcal{F} denotes the Fourier transform, and $\tilde{\mathbf{x}}$ is the signal in the spectral domain. The latter results from the orthogonality of the eigenvectors. Therefore, the Fourier transform is achieved by projecting the signal onto the transpose of the eigenvector matrix while the inverse transform is obtained by projecting onto the eigenvectors per se. The graph convolution is introduced in the next section.

4.2. Spectral Graph Convolution

Having defined the graph Fourier transform, one may immediately define the spectral graph convolution with the help of the convolution theorem. (Bronstein et al., 2017). The standard convolution is defined as

$$(\mathbf{f} \star \mathbf{g})(x) = \int_{\Omega} \mathbf{f}(x - x')\mathbf{g}(x')dx' \quad (21)$$

According to the latter, the convolution of a signal $\mathbf{x} \in \mathbb{R}^n$ with a parametric filter (g_{θ}) is defined as

$$\mathbf{x} \star_G g_{\theta} = \mathcal{F}^{-1}[\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(g_{\theta})] \quad (22)$$

Where $\theta \in \mathbb{R}^n$ is a parametrization of the filter and \odot is the element-wise or Hadamard product. As stated in the previous section, the Fourier transform and its inverse may be expressed in terms of the eigenvectors of the Laplacian matrix. Therefore, the spectral graph convolution becomes:

$$\mathbf{x} \star_G g_\theta = \Phi(\Phi^T \mathbf{x}) \odot (\Phi^T g_\theta) \quad (23)$$

Because of the orthogonality of the eigenvectors, this equation may be further simplified:

$$\mathbf{y} = \mathbf{x} \star_G g_\theta = g_\theta(\mathbf{L})\mathbf{x} = \Phi g_\theta(\Lambda) \Phi^T \mathbf{x} = \underbrace{\Phi \text{diag}(g_\theta) \Phi^T}_{\tilde{G}} \quad (24)$$

where the kernel is given by $g_\theta(\Lambda) = \text{diag}(g_\theta(\lambda))$. The filters are parametrized in terms of a truncated power series over the eigenvalue matrix. Because this matrix is diagonal, only the diagonal element needs to be exponentiated, thus reducing the complexity of the calculation from $\mathcal{O}(N^2)$, for a dense matrix, to $\mathcal{O}(N)$:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \quad (25)$$

where $\theta_k \in \mathbb{R}^K$ are the filter's parameters (Defferrard et al., 2016). Most filters are not localized because their spectrum has an infinite expansion (Defferrard et al., 2016). This is in contrast with the standard convolution for which the filters associated with CNN are always localized. Nonetheless, localization may be achieved if the filters are expressed in terms of Chebyshev polynomials of the first kind (Hammond et al., 2011; Wu et al., 2019):

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}) \quad (26)$$

where $\tilde{\Lambda} = \frac{2\Lambda}{\lambda_{max}} - \mathbf{I}_n$ is the rescaled eigenvalue matrix and $T_k(\tilde{\Lambda})$ is the Chebyshev polynomial of order k . A truncated development of order $K - 1$ corresponds to a filter that spans a K -ring neighborhood: from a 1-ring up to a K -ring. Each polynomial corresponds to a particular neighborhood: T_1 corresponds to a 1-ring neighborhood, T_2 corresponds to a 2-ring neighborhood while T_k corresponds to a k -ring neighborhood away (Kipf & Welling, 2016; Hammond et al., 2011). The Chebyshev polynomial may be defined by a recurrence relation, thus reducing the computational complexity. Indeed, given:

$$\bar{\mathbf{x}}_k = T_k(\tilde{\mathbf{L}})\mathbf{x} \in \mathbb{R}^n \quad (27)$$

where the scaled Laplacian is defined as

$$\tilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{max}} - \mathbf{I}_n \quad (28)$$

the recurrence relation is given by

$$\bar{\mathbf{x}}_k = 2\tilde{\mathbf{L}}\bar{\mathbf{x}}_{k-1} - \bar{\mathbf{x}}_{k-2} \quad (29)$$

With $\bar{\mathbf{x}}_0 = \mathbf{x}$ and $\bar{\mathbf{x}}_1 = \tilde{\mathbf{L}}\mathbf{x}$. As a result, the complexity associated with the filtering operation $\mathbf{y} = g_\theta(\mathbf{L})\mathbf{x} = [\bar{x}_0, \bar{x}_1, \dots, \bar{x}_{K-1}]$ is $\mathcal{O}(K|E)$, where $|E|$ is the number of edges.

Our graph convolutional network is described in the next section.

4.3. Graph Convolution Network

The spectral graph convolution in the non-Euclidean domain is obtained by applying the graph Fourier transform and the convolution theorem to both the input signal and the convolving filter (Bruna et al., 2013),

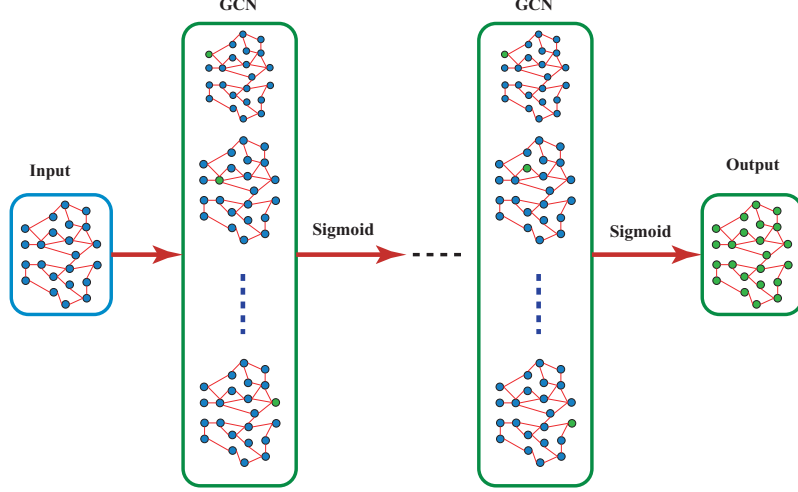


Figure 4: Graph Convolutional Neural Network

The graph convolution extracts underlying local information by collecting the node information in a local neighborhood whose extension is determined by the order of the Chebyshev polynomial. In order to extract multi-scale substructure features, multiple graph convolution layers may be stacked as illustrated in Fig. 4. The propagation rule for the multi-layer Graph Convolutional Network (GCN) is given by

$$\mathbf{f}_i^{l+1} = \sigma \left(\sum_{j=1}^p \Phi \hat{\mathbf{G}}_{i,j} \Phi^T \mathbf{f}_i^l \right), \quad i = 1, \dots, q, \quad j = 1, \dots, p \quad (30)$$

Where σ denotes the nonlinear activation function (l) and $\hat{\mathbf{G}} = \text{diag}(g_\theta(\lambda))$. The number of features is indicated by (q) while the number of assets is given by (p).

Algorithm 1 Implementation of Graph Convolutional Network

Input: $\mathbf{x}_t = [v_1, v_2, v_3]^T$

Output: Output feature vector $\leftarrow \mathbf{F} = [\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3]^T = [\mathbf{f}_1^{l+1}, \mathbf{f}_2^{l+1}, \mathbf{f}_3^{l+1}]^T$

Data: v_1, v_2, v_3 : Extracted features from restricted stacked autoencoder for each asset

Data: $\mathbf{f}_i^{l+1}, \sigma$: Transformed feature of layer $l + 1$, Activation function [Sigmoid]

1 Initialization:

1. **Calculation of the graph Laplacian:**

$$\mathbf{L} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2}$$

2. **Eigendecomposition of the graph Laplacian:**

$$\mathbf{L} = \Phi \Lambda \Phi^T$$

$$\tilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{max}} - \mathbf{I}_n$$

3. **Chebyshev polynomial (k, \mathbf{x}_t):**

$$\hat{\mathbf{x}}_0 = \mathbf{x}_t$$

if $k == 1$ **then**

$$\hat{\mathbf{x}}_1 = \tilde{\mathbf{L}}\mathbf{x}$$

else

$$\hat{\mathbf{x}}_k = \tilde{\mathbf{L}}\hat{\mathbf{x}}_{k-1} - \hat{\mathbf{x}}_{k-2}$$

4. **Kernel definition:**

$$\Phi g_\theta(\Lambda) \Phi^T = \underbrace{\Phi \text{diag}(\hat{g}_\theta) \Phi^T}_{\hat{\mathbf{G}}}$$

5. **Graph Convolution:**

$$\mathbf{f}_i^{l+1} = \sigma \left(\sum_{j=1}^p \Phi \hat{\mathbf{G}}_{i,j} \Phi^T \mathbf{f}_i^l \right), \quad i = 1, \dots, q, \quad j = 1, \dots, p$$

Algorithm 1 describes the graph convolutional network architecture. The output of the GCN is employed to train the actor–critic algorithm. The reinforcement learning approach is explained in detail in the following section.

5. Deep Reinforcement Learning Framework

Deep reinforcement learning algorithms may be divided into three categories: critic-only, actor-only, and actor–critic methods. The critic-only approaches include, among others, Q-learning (Watkins & Dayan, 1992; Bu et al., 2008) and SARSA (Zhao et al., 2016); the latter was employed in DeepBreath framework (Soleymani & Paquet, 2020). Critic-only methods rely on a state-action value function without explicitly defining a function over the policy space. Q-learning is an off-policy RL method in which the generated actions may be unrelated to the improvement of the policy. On the other hand, SARSA is an on-policy RL algorithm that seeks to improve the policy that generates the current actions (Andrew, 1999). On the other hand, actor-only methods optimize the cost over parameter space using a policy gradient approach. The actor-only methods converge much faster than critic-only methods (Grondman et al., 2012). The actions generated by an actor-only method span a continuum, but the variance associated with the gradient is large, which results in a slower learning rate (Konda & Tsitsiklis, 2000; Sutton et al., 2000). Finally, the actor–critic algorithm was developed in order to combine the advantages of both the actor-only and critic-only methods. Pseudo code for our actor–critic algorithm appears in section 5.1. Reinforcement learning methods apply a sequence of actions taken by an agent, to ensure that the expected cumulative reward be optimum. In our framework, the actions correspond to the weight vector associated with the portfolio.

$$\mathbf{a}_t = \mathbf{w}_t. \quad (31)$$

While the reward aims to maximize the expected return on investment. When considering the relation between the return rate Eq. 6a and the transaction factor Eq. 10, it may be concluded that the current action depends on the previous one. The state of the portfolio is obtained by concatenating the internal and external states: the former being the portfolio weight vector at time $t - 1$ (\mathbf{w}_{t-1}) while the latter is the current feature tensor X_t :

$$\mathbf{s}_t = [\mathbf{X}_t, \mathbf{w}_{t-1}]^T. \quad (32)$$

The performance of each action is evaluated based on the achieved reward, aiming to increase the reward over a finite investment horizon ($t_f + 1$). The reward is determined by the logarithmic accumulated return \mathcal{R} : Eq. 6b, 7 and 10:

$$\begin{aligned} \mathcal{R}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) &= \frac{1}{t_f} \cdot \ln \left(\frac{P_f}{P_0} \right) \\ &= \frac{1}{t_f} \cdot \sum_{t=1}^{t_f+1} \ln(\mu_t \cdot \mathbf{Y}_t \cdot \mathbf{w}_{t-1}) = \frac{1}{t_f} \cdot \sum_{t=1}^{t_f+1} R_t. \end{aligned} \quad (33)$$

The actor–critic algorithm used to learn the policy is described in the next section.

5.1. Actor–Critic Architecture

As the name suggests, the actor–critic algorithms–consists of two main components: the actor and the critic. The actor undertakes a sequence of actions following a policy while the critic evaluates the policy by assigning a performance index, called the value function, to each of the actor’s actions. The performance is evaluated using temporal differences (TD) (Sutton, 1988). The critic approximates and updates the value function, which indicates the direction in which the actor’s policy parameters should be updated to improve performance. In this method, the policy update is derived directly from the value function, while the value function enforces the direction of the policy gradient at each time step (Grondman et al., 2012). Actor–critic algorithms are capable of generating continuous actions while avoiding large policy gradient variance.

In this work, an actor–critic algorithm is employed in order to find the optimal policy for a higher return on investment. The actor undertakes the actions while the critic evaluates the state-value function as a success metric for the actions. The actor updates the policy in the direction suggested by the critic. The actor consists of a deep convolutional neural network, as illustrated in Fig. 5. Convolutional neural networks are known for their ability to learn complex patterns and efficiently implement policies (Jiang et al., 2017). The output of the actor is the learned policy, which determines the portfolio allocation. The network is trained in two stages: initially, offline learning is

performed based on a historical dataset using online stochastic batching (Jiang et al., 2017), leading to the offline policy. Then, as new data become available, the policy is updated online. The architecture of the actor is shown in Algorithm 2. The input tensor of this network consists of the high-level features learned from the restricted stacked autoencoder. The tensor consists of three (3) channels, with which three (3) matrices are associated.

The number of rows in each matrix is determined by the number of assets (m), while the number of columns corresponds to the size of the trading window (n). The initial convolution is performed on the input tensor with a kernel of 1×3 size, and then a $Tanh$ activation function is applied. The second convolutional layer has a kernel of size $1 \times n$ aiming to obtain a vector of size m . The third layer is connected to the fourth layer through convolution with a kernel of 1×1 size (obtaining one main channel) and a $Tanh$ activation function. The third layer consists of the previous actions (portfolio vector), i.e., the current action is computed based on both the current state and previous actions. The cash bias is added to the acquired action vector in the fourth layer, generating a vector of size $m + 1$. The last layer normalizes the action vector using a Softmax function, resulting in a new portfolio weight distribution. All of the dimensions were determined by inspection.

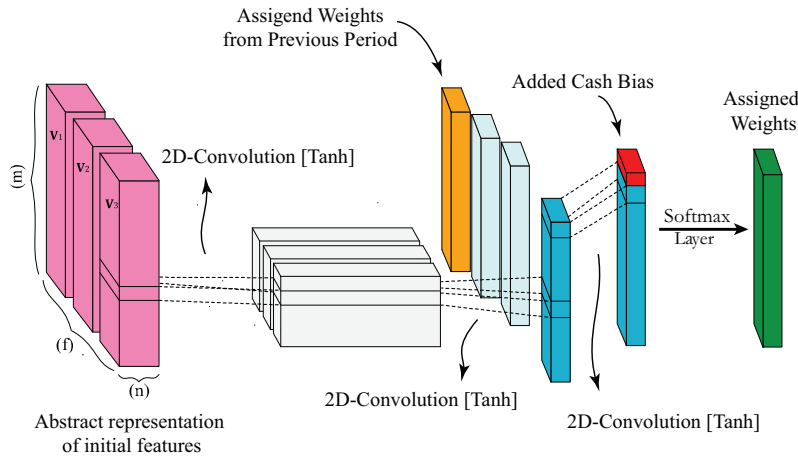


Figure 5: Architecture of actor

Similarly, the architecture of the critic is described in Fig. 6, and the corresponding pseudocode may be found in Algorithm 3. The input consists of the current feature tensor \mathbf{X}_t . The size of the kernels for the first and second convolutional layers, is 1×1 . This is followed by a ReLU activation function. The third convolutional layer has a kernel of size $1 \times m$ and ReLU activation function. Finally, the approximated value function is obtained by means of a densely connected layer.

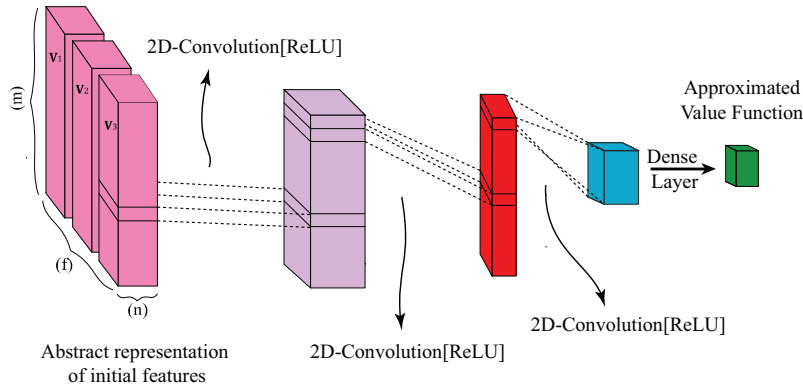


Figure 6: Critic architecture based on deep convolutional neural network

Algorithm 2 Implementation of Actor Algorithm

Input: $\mathbf{X}_t = [\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3]^\top$ **Output:** Distributed Portfolio Weights $\leftarrow \mathbf{W}_{ptf} = [\mathbf{w}_C, \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m]^\top$ **Data:** f, m : Number of Features, Number of Assets**1 Initialization:****while** $i \leq n$ **do****1. First Convolution:**

Activation Function: Tanh

Kernel size $\rightarrow (1,3)$ Kernel depth $\rightarrow f$ **2. Second Convolution:**

Activation Function: Tanh

Kernel size $\rightarrow (1,n)$ Kernel depth $\rightarrow f$ **3. Third Convolution:**

Concatenate weights for previous and current time steps (Weight Matrix).

 $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_i]^\top$

Activation Function: Tanh

Kernel size $\rightarrow (1,1)$ Kernel depth $\rightarrow f + 1$ **4. Adding Cash bias:**

Concatenate Cash Bias and computed Weight Matrix

5. Softmax Layer:

Distributed Portfolio Weights

The actor is implemented with convolutional neural networks, including three convolutional layers and a final Softmax layer that generates the weights distribution vector. The current action at time t is a function of the action at time $t - 1$. The loss function of this network is evaluated based on the log-normal distribution of the policy π_θ and the temporal difference error (TD error) calculated by the critic network. The actor neural network follows a policy gradient approach, which updates the weight using a one-step TD error from Eq. 35.

Algorithm 3 Implementation of Critic Algorithm

Input: $\mathbf{X}_t = [\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3]^\top$ **Output:** Approximated value function V_v **Data:** f, m, n : Number of Features, Number of Assets, Trading Window Length**1 Initialization:****while** $i \leq n$ **do****1. First Convolution:**

Activation Function: ReLU

Kernel size $\rightarrow (1,1)$ Kernel depth $\rightarrow f$ **2. Second Convolution:**

Activation Function: ReLU

Kernel size $\rightarrow (1,1)$ Kernel depth $\rightarrow n$ **3. Third Convolution:**

Activation Function: ReLU

Kernel size $\rightarrow (1,m)$ Kernel depth $\rightarrow 1$ **4. Dense Layer**Value (V_v) Approximation**5. Training**

Compute the TD error using Eq. 35.

Minimize the loss

Compute loss using Eq. 34

The critic consists of three convolutional neural networks. The critic network is updated aiming to minimize, in Eq. 34, the mean square error (MSE) between the approximated value function V_v and target value V^π .

$$MSE = \|V^\pi - V_v\|^2 \quad (34)$$

The weights of the critic network are updated based on the gradient descent method, as shown in Algorithm 3. The actor-critic framework is detailed in Algorithm 4.

$$\delta_v = r(\mathbf{s}, \mathbf{a}) + \gamma V_v(\mathbf{s}') - V_v(\mathbf{s}) \quad (35)$$

where $r(\mathbf{s}, \mathbf{a})$ denotes immediate reward at state \mathbf{s} resulting from action \mathbf{a} . The trade-off in between immediate and long term strategy is established by γ . The weights of the critic network are updated based on the gradient descent method as shown in Algorithm 3. The actor-critic framework is detailed in Algorithm 4.

Algorithm 4 Actor-Critic Algorithm

Input: Initial network weights: $\theta_{a,init}, w_{c,init}$ **Data:** $\alpha_a, \alpha_c \in [0, 1]$: Actor and critic learning rate**Data:** θ_a, w_c : Actor and critic neural network weights**Data:** N_t : Number of iterations**1 Initialization:** $iteration = 0$ $\pi_\theta(\mathbf{s}, \mathbf{a}) \in \mathcal{A}$ **while** $iteration \leq N_t$ **do****2** $\Delta_w = \alpha_c \delta_v \nabla_w V(\mathbf{s})$ $w_c \leftarrow w_c + \Delta_w$ $\Delta_\theta = \alpha_a \delta_v \nabla_\theta \log \pi_\theta(\mathbf{s}, \mathbf{a})$ $\theta_a \leftarrow \theta_a + \Delta_\theta$ *Update state and action:* $\mathbf{s} \leftarrow \mathbf{s}'$ $\mathbf{a} \leftarrow \mathbf{a}'$

The actor-critic algorithm uses the Adam optimization method (Kingma & Ba, 2015) to update the weights for both the actor and the critic. The actor and the critic act alternately. At first, the weights associated with the networks are initialized randomly, and an initial admissible policy π_θ is calculated. The portfolio state is determined with respect to the initial action, and the value function is approximated. Then, the weights of the critic network are updated. Afterward, the weights of the actor network are updated using the TD error measured by the critic network. Finally, the state (\mathbf{s}) and action (\mathbf{a}) are updated. In other words, at each step, the actor network yields an action (\mathbf{a}) at state (\mathbf{s}), following the direction enforced by the critic network, in order to minimize the value function (V_v).

6. Online Learning

The proposed framework is trained using historical data (prior to current calendar time) and the current flow of data pertaining to offline and online learning, respectively. The offline learning is performed using online stochastic batching, described in (Soleymani & Paquet, 2020). Financial data in the stock market may be assimilated into time series. As a result, unanticipated changes in their underlying distribution may occur, a phenomenon known as concept drift (Gama et al., 2014). These may be generated by exogenous factors such as panic reactions (e.g. the Covid-19 pandemic) (Ashraf, 2020; Zhang et al., 2020), and political disputes (e.g. US-China trade war) (Cavalcante & Oliveira, 2015). Such factors may profoundly affect the actions taken by investors. In this work, concept drift is addressed using a passive detection approach where the agent is first trained based on historical data, and then the learned policy is continuously updated as new data arrive, thus allowing for concept drift without discarding useful knowledge (Soleymani & Paquet, 2020).

The motivation of the present approach is twofold: extreme events are very often recurrent, which means that a solution to a current financial crisis may be brought from past crises (Hu et al., 2015). Secondly, the passive detection approach discards historical patterns that are not repeated after a certain amount of time (Gama et al., 2004), thus preventing any unstable, short-term investment strategies based on outliers. All of the neural networks in our portfolio management framework (RSAE, GCN, actor-critic), are first trained offline on historical data. Then, the weights are updated online as new data become available. The online learning is managed by a buffer, as shown in Fig. 7. The buffer stores the financial data from the last ten days, along with the current business day. Then the stored set containing data from eleven days is used to train and update the networks. At the end of the business transaction (current day), the current financial data are added to the offline dataset.

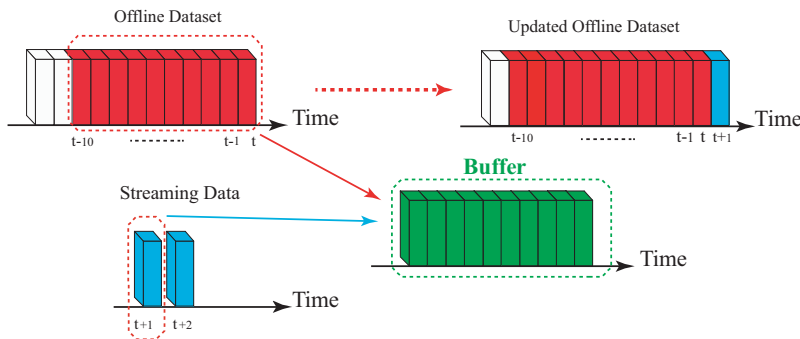


Figure 7: Structure of the buffer that stores data from the offline dataset as well as the flow of online data for online learning

7. Experimental Results

The assets forming our portfolio appear in Table 2. This dataset consists of various financial time series which cover a period ranging from January 2, 2002, up to March 24, 2020; details about each time series may be found in Table 3. In reinforcement learning, actions are taken to minimize the risk while increasing the return on investment. Therefore, to further investigate the risk involved with the investment policy, the performance of the proposed framework is evaluated using three of the most celebrated risk evaluation metrics (Bessler et al., 2017): namely maximum drawdown (MDD), Sharpe ratio (S_r), and Conditional-Value-at-Risk (CVaR) (Rockafellar et al., 2000). The maximum drawdown metric was first proposed by (Magdon-Ismail & Atiya, 2004). This metric is defined as the difference in between the maximum peak value P and the minimum value L reached by the portfolio

for a certain period of time before the next peak occurs. Technically, this indicator measures the amplitude of a loss over a period of time. A low maximum drawdown indicates a small loss.

$$MDD = \frac{P - L}{P}. \quad (36)$$

The Sharpe ratio was introduced by (Sharpe, 1994). It measures the volatility of investment as a ratio between the expected return on investment and the risk:

$$S_r = \frac{E[R_f - R_b]}{\sigma_d}. \quad (37)$$

where σ_d is the standard deviation associated with the asset excess return (or volatility) where $(E[R_f - R_a])$ is the expected differential return. A high return on investment is achieved when the Shape ratio is greater than one. In other words, this metric helps investors to better identify the returns associated with high-risk actions.

In order to further analyze the risk associated with portfolio optimization, risk assessment criteria called Value-at-Risk (VaR) and Conditional-Value-at-Risk (CVaR) are adopted here. The former measures and control the risk with respect to percentiles of the loss distribution, while later estimates the tail risk of a portfolio (Alexander et al., 2006). Nonetheless, the use of VaR is limited due to lack of subadditivity and being non-convex and non-smooth with multiple local minima (Acerbi, 2002). On the other hand, the CVaR renders a convex optimization problem (Cormuejols & Tütüncü, 2006).

$$CVaR_\alpha(X) = \mathbb{E}[X|X \geq VaR_\alpha(X)] \quad (38)$$

The historical training set consists of four different time series as shown in Table. 3. Each training set spans a different period, allowing our system to learn from various historical contexts. Each training batch covers 90 consecutive days. Initially, the portfolio consists only of cash, so the portfolio composition is determined entirely by the agent from the start. As mentioned earlier, the assets forming the portfolio appear in Table. 2. These assets were selected in order to reflect various segments of the economy, such as the pharmaceuticals industry, financial services, manufacturing, healthcare, energy, and consumer services. This portfolio allows us to further investigate the interrelation among financial instruments that may be perceived as statistically independent at first sight.

Table 2: Composition of our Portfolio for the period in between January 2, 2002, and February 10, 2020.

Sector	Name
<i>Technology</i>	Apple.Inc (AAPL)
	Cisco Systems Inc. (CSCO)
	Intel Corporation (INTC)
	Oracle Corporation (ORCL)
	Microsoft Corporation (MSFT)
	International Business Machines Corporation (IBM)
	Honeywell International Inc. (HON)
Verizon Communications Inc. (VZ)	
<i>Financial Services</i>	Manulife Financial Corporation (MFC)
	JPMorgan Chase & Co. (JPM)
	Bank of America Corporation (BAC)
	The Toronto-Dominion Bank (TD)
<i>Industries</i>	3M Company (MMM)
	Caterpillar Inc. (CAT)
	The Boeing Company (BA)
	General Electric Company (GE)
<i>Consumer Defensive</i>	Walmart Inc. (WMT)
	The Coca-Cola Company (KO)
<i>Consumer Cyclical</i>	The Home Depot, Inc. (HD)
	Amazon.com, Inc. (AMZN)
<i>Healthcare</i>	Johnson & Johnson (JNJ)
	Merck & Co., Inc. (MRK)
	Pfizer Inc. (PFE)
	Gilead Sciences, Inc. (GILD)
<i>Energy</i>	Enbridge Inc. (ENB)
	Chevron Corporation (CVX)
	BP p.l.c. (BP)
	Royal Dutch Shell plc (RDSB.L)

Table 3: Training and test sets

ID	Training Set	Test Set
<i>Test 1</i>	2002-01-02 to 2009-04-16	2010-03-15 to 2010-07-21
<i>Test 2</i>	2002-01-02 to 2012-12-06	2013-11-04 to 2014-03-14
<i>Test 3</i>	2002-01-02 to 2016-08-01	2017-06-28 to 2017-11-02
<i>Test 4</i>	2002-01-02 to 2018-10-19	2019-06-09 to 2019-10-16
<i>Test 5</i>	2002-01-02 to 2019-06-23	2019-11-12 to 2020-03-24

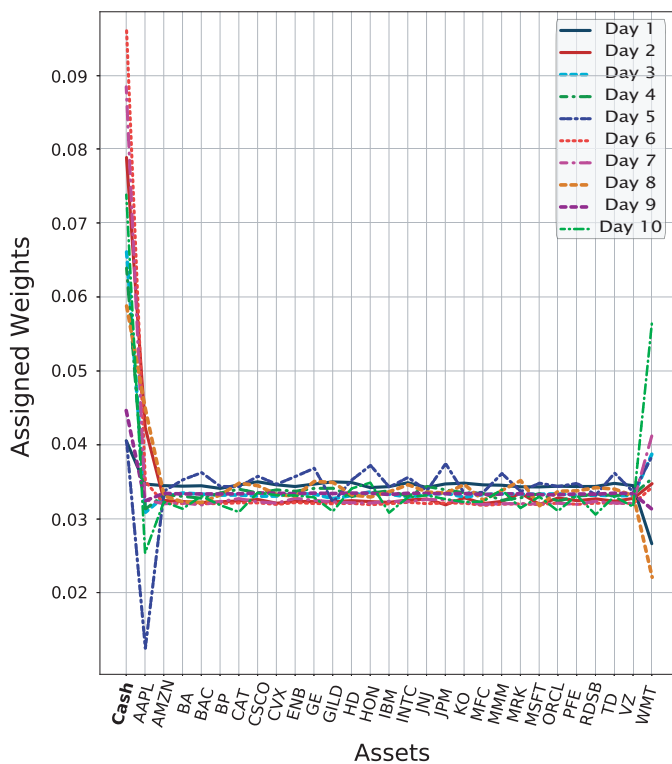


Figure 8: Evolution of the portfolio weights during the first 10 days of the online learning process.

Our online learning results are shown in Fig. 8, where the pre-trained model from the last training set is employed, and the weights for the online model are updated as new data becomes available (for each trading day). The learning is conducted over 10 trading days from 2020-03-24 to 2020-04-06. Fig. 8 illustrates the weight fluctuations during the online learning process. Initially, the lowest weight corresponds to WMT, while the rest of the assets are uniformly distributed. Then, as the process unfolds, the weights associated with assets such as AAPL and WMT are subject to great variations, while others, such as ORCL, PFE, and RDSB, are not. The most stable assets, in terms of their weights in the portfolio, are employed by the agent to achieve a stable return on investment by hedging the risk. More volatile assets, in terms of their weights, are exploited by the agent for their leverage effect (Bouchaud et al., 2001).

Fig. 9 illustrates the weight distribution for the four test sets, as defined in Table 3, after respectively 30, 60, and 90 days of investment. One may notice that the weights are more equally distributed for the last testing set. The behavior may be explained by the fact that the agent aims to mitigate risk by distributing the funds among the various assets. The resulting portfolio values are reported in Fig. 10. The portfolio value for the first test set is highlighted in blue. During the first 23 days, the portfolio value increases by 23.8% and then, after 60 days, climbs gradually to 30.33%. Then, the portfolio value remains stable while, during the last five days, the growth decreases to 26.23% after 90 days of trading. The evolution of the portfolio for the second test set consistently increases, showing a growth of 79.57% after 90 days. As for the third test, the growth reaches 83.97% after 90 days, the highest among all test sets. Finally, the fourth test set achieves 68.8% growth after being 68 days on the market before gradually dropping to 60.81% after 90 trading days. From Fig. 10, one may conclude that the proposed framework is particularly suitable for medium-term investments.

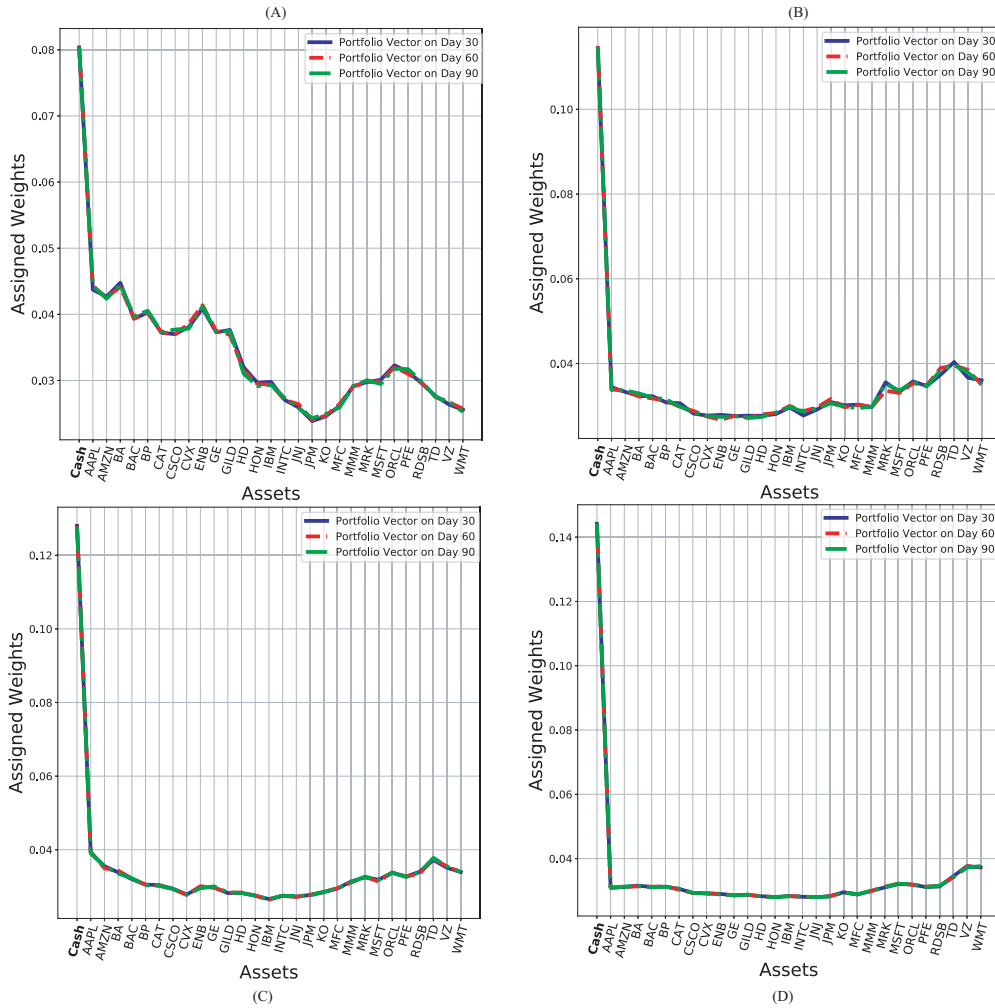


Figure 9: Portfolio weight distributions at the end of the investment horizon for four test sets.

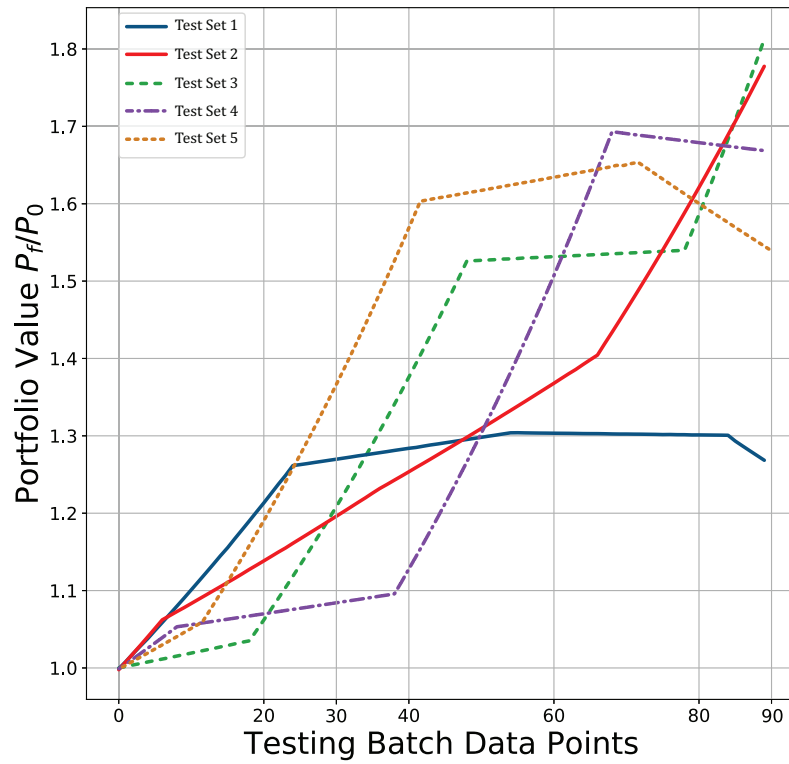


Figure 10: Relative portfolio values after respectively 30, 60, and 90 days of trading for all five test sets.

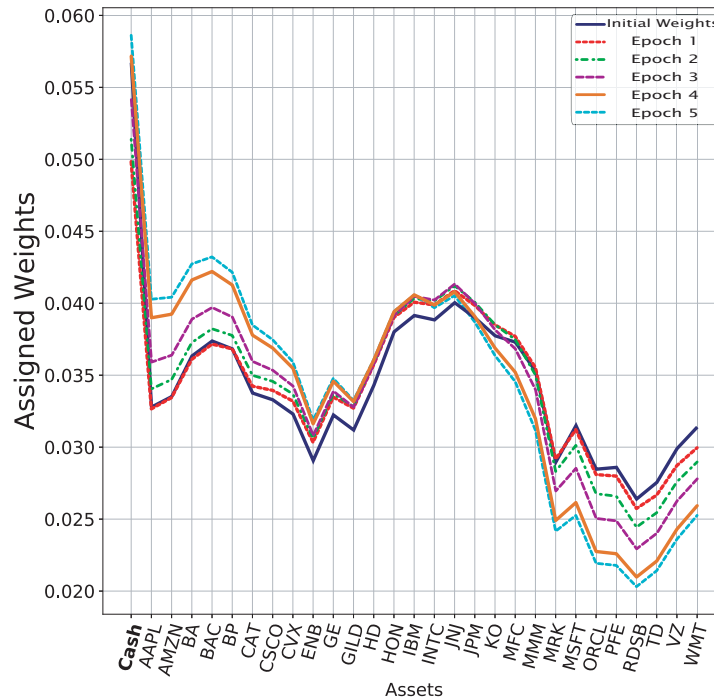


Figure 11: Final portfolio weights distributions for the fifth training set.

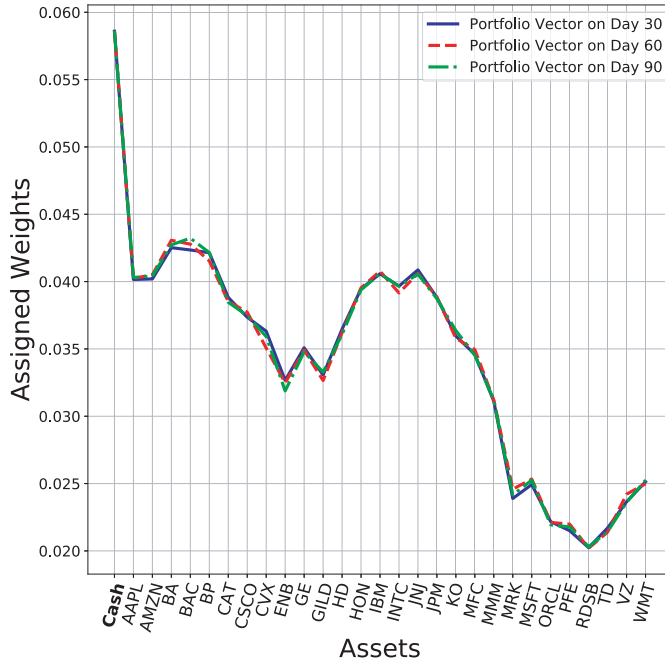


Figure 12: Final portfolio weight distributions for the fifth test set.

Table 4: Maximum drawdown (MDD) and Sharpe ratio (S_r) for DeepPocket, the Dow Jones Industrial (DJI), Euro Stoxx 50, Nasdaq, and S&P500 for all five test sets.

ID	Investment Duration	MDD (%) Algorithm	S_r Algorithm	S_r DJI	S_r FEZ	S_r Nasdaq	S_r S&P500
<i>Test Set 1</i>	30 Days	21.16	2.89	2.101	-3.80	1.766	1.66
	60 Days	23.31	3.11	-1.44	-2.133	-1.62	-1.55
	90 Days	23.31	2.95	-1.458	-1.722	-1.362	-1.42
<i>Test Set 2</i>	30 Days	16.096	3.41	2.752	-1.151	1.746	1.66
	60 Days	26.64	3.507	-1.518	-2.814	0.222	-1.65
	90 Days	43.74	3.84	0.949	-0.418	2.261	1.83
<i>Test Set 3</i>	30 Days	16.18	3.18	0.755	-1.94	-0.3914	-1.070
	60 Days	34.68	2.53	2.23	-0.987	0.755	1.27
	90 Days	44.83	3.39	3.018	1.65	2.819	2.43
<i>Test Set 4</i>	30 Days	7.71	3.24	1.75	1.93	1.951	2.031
	60 Days	32.70	3.67	-0.520	-2.518	-0.0521	-0.285
	90 Days	40.93	2.47	1.6134	0.114	0.981	1.42
<i>Test set 5</i>	30 Days	26.34	3.107	2.51	1.65	3.300	3.014
	60 Days	38.794	2.64	2.59	0.757	3.265	3.03
	90 Days	39.53	2.301	-3.78	-3.678	-2.314	-3.21

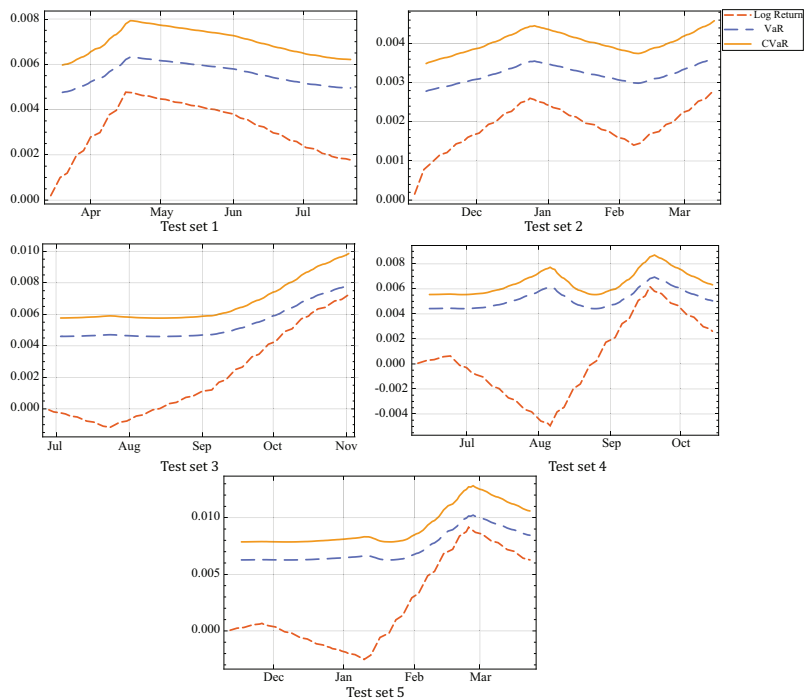


Figure 13: Risk assessment using CVaR and VaR criteria for all five test sets

Our results are summarized in Table 4. With the exception of Test Set 1, the Sharpe ratio is either very good or excellent for all four remaining test sets. In general, it is recognized that a Sharpe ratio below one is sub-optimal; a ratio between one and two is fair, and a ratio between two and three is considered very good. Any ratio over three is considered excellent (Maverick, 2019). In general, online training over a larger time window results in a more efficient policy in terms of return on investment. Our frameworks seems particularly adapted for medium-term investments as the value of the portfolio usually peaks after 80 to 90 trading days.

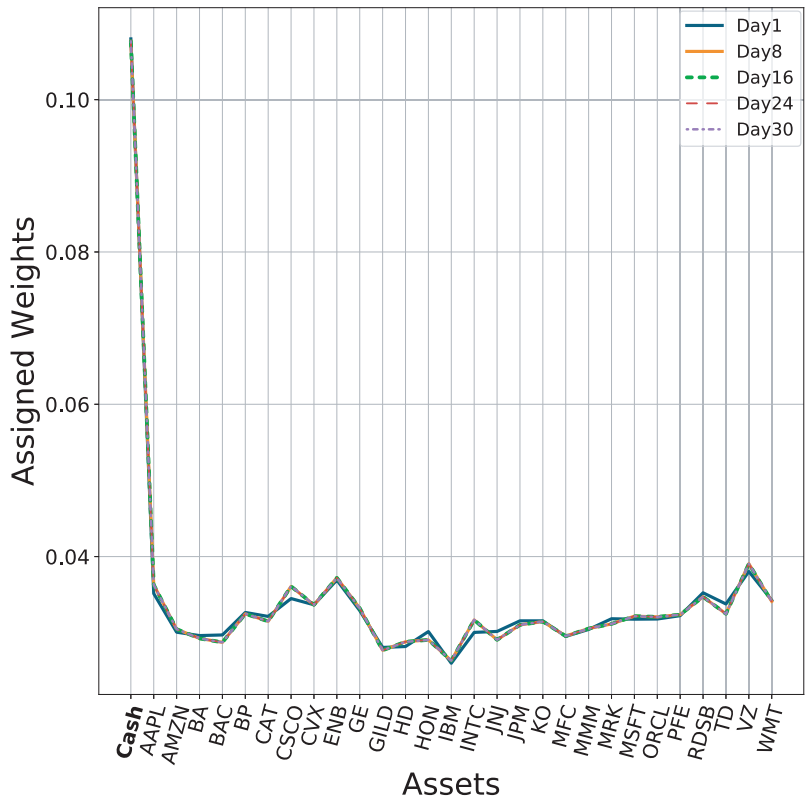


Figure 14: Evolution of the portfolio weights over a period of 30 days following the Covid-19 crisis.

Table 5: Portfolio weights distribution over a period of 30 days following the Covid-19 crisis(from Fig. 14)

ID	Day 1	Day 8	Day 16	Day 24	Day 30
Cash	0.10905	0.10878	0.10845	0.10807	0.10735
AAPL	0.0349	0.0361	0.0362	0.0365	0.037
AMZN	0.0299	0.0305	0.03052	0.03055	0.03055
BA	0.0296	0.0292	0.02918	0.02918	0.02918
BAC	0.0297	0.0286	0.0287	0.0288	0.02885
BP	0.0327	0.0322	0.0324	0.0322	0.0322
CAT	0.0322	0.03164	0.0315	0.03185	0.03185
CSCO	0.0344	0.03616	0.03605	0.03602	0.03602
CVX	0.03368	0.03367	0.03367	0.03367	0.03367
ENB	0.0367	0.037	0.037	0.03724	0.03724
GE	0.0328	0.033	0.0328	0.0326	0.0326
GILD	0.0283	0.0279	0.0278	0.0279	0.02792
HD	0.0281	0.0288	0.0291	0.0292	0.0292
HON	0.0301	0.0291	0.029	0.0289	0.0289
IBM	0.0259	0.0261	0.0262	0.0266	0.0266
INTC	0.0299	0.0317	0.0316	0.03158	0.03158
JNJ	0.0302	0.02906	0.02908	0.02915	0.02915
JPM	0.0316	0.03097	0.031	0.0312	0.0312
KO	0.0315	0.03148	0.03143	0.03141	0.03141
MFC	0.0297	0.0297	0.0297	0.0295	0.0295
MMM	0.0303	0.03046	0.03058	0.03062	0.03062
MRK	0.0318	0.03125	0.0313	0.0313	0.0313
MSFT	0.03172	0.0321	0.03211	0.0322	0.0322
ORCL	0.0321	0.03208	0.03208	0.03207	0.03207
PFE	0.03225	0.03235	0.0325	0.0321	0.0321
RDSB	0.0351	0.0348	0.03475	0.03471	0.03471
TD	0.0338	0.0324	0.0325	0.0326	0.03275
VZ	0.0378	0.039	0.0389	0.0389	0.0389
WMT	0.0342	0.0339	0.0339	0.03338	0.03338

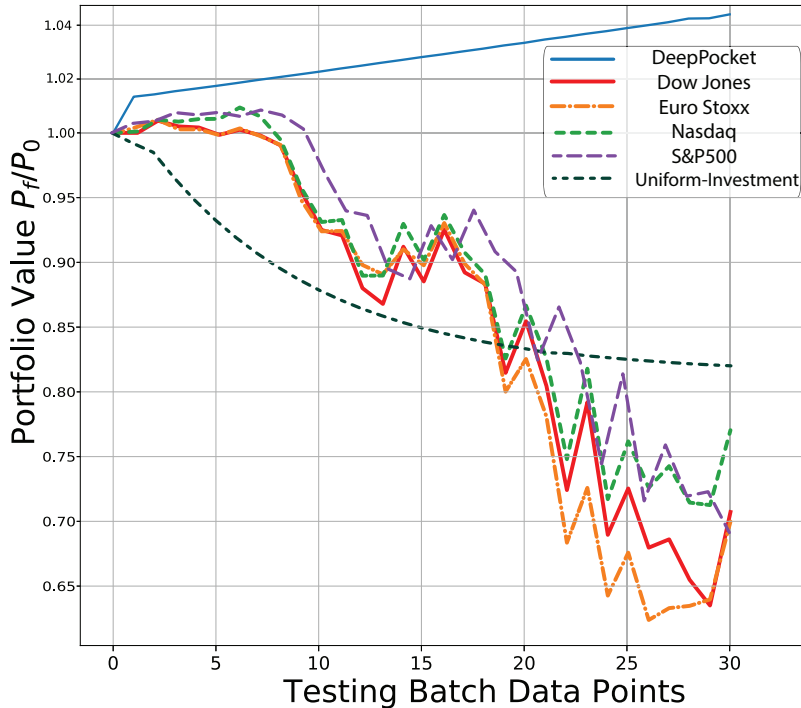


Figure 15: Evolution of the portfolio value over a period of 30 days following the Covid-19 crisis.

In order to test the robustness of our framework, DeepPocket managed the portfolio in the depths of the coronavirus crisis (Covid-19). The investments were performed from 11 February 2020, 7 days before the crisis, until 23 March 2020, while the stock market was witnessing record falls. The weight distribution during this time period is shown in Fig. 14 and Table. 5. The evolution of the portfolio is illustrated in Fig. 15. The objective of this test was to evaluate the performance of the agent in unprecedented circumstances as the stock market was entering a bear market (Barsky & De Long, 1990). As shown in Fig. 15, despite the debacle in worldwide financial markets, DeepPocket managed to increase the value of the portfolio by 4.46%. This is even more remarkable when considering that the agent was trained online on a bull market during a sharp rise. This test shows the agent’s adaptation capacity under the worst possible conditions. During this period, some assets were more affected than others: indeed, while IBM, GILD, JPM, and HON were in sharp decline, AMZN and WMT were rapidly recovering.

In comparison, as illustrated by Fig. 15, had one invested evenly across these assets, the portfolio value would have shrunk by 17.83%. Moreover, during the same period, the Dow Jones Industrial (DJI) lost 29.36% of its value. Therefore, DeepPocket clearly outperforms uniform-investment and benchmark indexes in times of crisis. Our framework has the ability to remember remote events because of, among other reasons, gradual concept drift (Gama et al., 2014). The fact that DeepPocket could manage the coronavirus crisis is to be found, in part, in such a long-term memory mechanism. Indeed, the offline training, which was performed on historical data, involved the early 2000’s recession where the market reached a low point in 2002 as well as the 2009 market crisis (Langdon et al., 2002; Farmer, 2012). The knowledge gained by DeepPocket over these two crisis periods was instrumental in the management of the coronavirus crisis.

Table 6: Return on investment for DeepPocket, the Dow Jones Industrial, Euro Stoxx 50, Nasdaq, and S&P500 for all five test sets.

ID	Investment Duration	ROI(%) DeepPocket	ROI(%) DJI	ROI(%) FEZ	ROI(%) Nasdaq	ROI(%) S&P500
<i>Test Set 1</i>	30 Days	26.98	3.28	-6.48	4.625	2.88
	60 Days	30.33	-6.98	-21.88	-8.608	-8.24
	90 Days	26.23	16.908	-15.45	-7.403	-4.94
<i>Test Set 2</i>	30 Days	19.77	1.509	-2.08	2.212	0.739
	60 Days	36.91	0.38	-2.28	4.249	0.829
	90 Days	79.57	12.79	0.467	7.844	5.14
<i>Test Set 3</i>	30 Days	20.85	1.814	-0.807	-0.281	-0.101
	60 Days	53.16	4.17	0.58	3.087	2.521
	90 Days	83.97	20.20	4.539	7.70	6.029
<i>Test Set 4</i>	30 Days	8.5	4.30	3.55	5.280	4.675
	60 Days	50.78	-0.78	-4.54	0.923	0.013
	90 Days	60.81	4.71	0.619	4.866	4.837
<i>Test set 5</i>	30 Days	37.66	2.97	2.616	5.772	4.417
	60 Days	63.53	5.095	1.584	12.478	7.797
	90 Days	55.36	-32.86	-35.018	-18.722	-27.522

In order to further assess the performance of the proposed framework, the return on investment of DeepBreath was compared with four benchmark portfolios namely the Dow Jones Industrial (DJI), the Euro Stoxx 50 ETF, Nasdaq, and S&P500 (Serletis & Rosenberg, 2009). Our results are reported in Table 6. For instance, for test set 5, our framework achieved a 37.66%, 63.53%, and 55.36% return on investment (ROI) over thirty, sixty, and ninety days, respectively. Meanwhile, for the same investment periods, the ROI was 2.97%, 5.095%, and -32.86% for DJI, and 2.616%, 1.584%, -35.018% for Euro Stoxx 50 ETF, 5.772%, 12.478%, -18.722% for Nasdaq and finally 4.417%, 7.797%, -27.522% for S&P500.

8. Discussion

DeepPocket was evaluated against five real-life datasets over three distinct investment periods including during the Covid-19 crisis, for which it clearly outperformed market indexes. The results obtained during the Covid-19 crisis are particularly remarkable as DeepPocket managed to generate a profit by exploiting profitable assets, such as Amazon and Facebook, in addition to exploiting offline knowledge such as what it learned from the 2008 global financial recession. This performance is in contrast with market indexes, which were collapsing during the same period. The reason is that our deep neural network learns the underlying rules, connections and predictable patterns directly from the data. The network adapts as new information becomes available, which allows the system to make better predictions based on insight acquired from previously analyzed information. The amount of complex information that it can process far surpasses largely human capacities. The network may detect anomalies such as unexpected spikes, drops, level shifts and trend changes. In contrast to traders, DeepPocket is not prone to such a high degree of personal biases, conflicts of interest, and emotional decisions. Indeed, as reported recently in ¹, robot-analysis systems, like DeepPocket, tend to outperform traditional investment strategies.

8.1. Conclusion

In this paper, we proposed DeepPocket, a portfolio management framework that aims to increase the expected return on investment while hedging the risk. This framework exploits the time-varying correlations between financial instruments. These correlations are represented in a graph whose nodes correspond to the financial instruments and whose edges correspond to pair-wise correlation functions between assets. DeepPocket consists of a restricted, stacked autoencoder (RSAE) for feature extraction, and an actor-critic reinforcement learning agent. The actor-critic framework consists of two convolutional networks in which the actor learns and enforces an investment policy, and the critic, in turn, evaluates the policy to establish the best course of action by constantly reallocating the various portfolio assets to maximize the expected return on investment. The agent is initially trained offline with

¹<https://www.bloomberg.com/news/articles/2020-02-11/robot-analysts-outwit-humans-in-study-of-profit-from-stock-calls>

online stochastic batching on historical data. As new data becomes available, the agent is trained online with a passive concept-drift approach to handle unexpected changes in the underlying data distribution.

The approach has many benefits for practitioners such as seeking investment opportunities, identifying market patterns, evaluating trading strategies, and automating workflows, just to mention a few. By finding predictable patterns, structures and trends, the system may provide valuable recommendations and insight to traders. Risk modelling and forecasting, as well as their accuracy, may be improved using the insight acquired from the data. Traditional risk models have demonstrated their ineffectiveness in coping with global financial crisis. As shown by our results, DeepPocket successfully managed such risks during the Covid-19 crisis: a characteristic which has direct repercussions for portfolio risk management and compliance.

DeepPocket assumes a liquid market which means that many buyers and sellers are available, while price variations remain relatively small. We plan to target this problem in a future work, in order to address situations in which buyers and sellers are in short supply. Besides portfolio allocation, we plan to have DeepPocket select the stocks directly in addition to developing deep learning models more suitable for long-term investment strategies.

References

- Acerbi, C. (2002). Spectral measures of risk: A coherent representation of subjective risk aversion. *Journal of Banking & Finance*, *26*, 1505–1518.
- Alexander, S., Coleman, T. F., & Li, Y. (2006). Minimizing cvar and var for a portfolio of derivatives. *Journal of Banking & Finance*, *30*, 583–605.
- Andrew, A. M. (1999). Reinforcement learning: An introduction by richard s. sutton and andrew g. barto, adaptive computation and machine learning series, mit press (bradford book), cambridge, mass., 1998, xviii+ 322 pp, isbn 0-262-19398-1,(hardback,£ 31.95). *Robotica*, *17*, 229–235.
- Angles, R., & Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys (CSUR)*, *40*, 1–39.
- Anthony, J. H. (1988). The interrelation of stock and options market trading-volume data. *The Journal of Finance*, *43*, 949–964.
- Arfaoui, M., & Rejeb, A. B. (2017). Oil, gold, us dollar and stock market interdependencies: a global analytical insight. *European Journal of Management and Business Economics*, .
- Ashraf, B. N. (2020). Stock markets’ reaction to covid-19: cases or fatalities? *Research in International Business and Finance*, (p. 101249).
- Barabási, A.-L. et al. (2016). *Network science*. Cambridge university press.
- Barsky, R. B., & De Long, J. B. (1990). Bull and bear markets in the twentieth century. *The Journal of Economic History*, *50*, 265–281.
- Bengio, Y., Goodfellow, I. J., & Courville, A. (2015). Deep learning. *Nature*, *521*, 436–444.
- Bessler, W., Opfer, H., & Wolff, D. (2017). Multi-asset portfolio optimization and out-of-sample performance: an evaluation of black–litterman, mean-variance, and naïve diversification approaches. *The European Journal of Finance*, *23*, 1–30.
- Bhanja, S., & Das, A. (2018). Impact of data normalization on deep neural network for time series forecasting. *arXiv preprint arXiv:1812.05519*, .
- Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., & Lee, M. (2009). Natural actor–critic algorithms. *Automatica*, *45*, 2471–2482.
- Bouchaud, J.-P., Matacz, A., & Potters, M. (2001). Leverage effect in financial markets: The retarded volatility model. *Physical review letters*, *87*, 228701.
- Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, *34*, 18–42.
- Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, .

- Bu, L., Babu, R., De Schutter, B. et al. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38, 156–172.
- Cavalcante, R. C., & Oliveira, A. L. (2015). An approach to handle concept drift in financial time series based on extreme learning machines and explicit drift detection. In *2015 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.
- Celikyurt, U., & Özekici, S. (2007). Multiperiod portfolio optimization models in stochastic markets using the mean–variance approach. *European Journal of Operational Research*, 179, 186–202.
- Cervelló-Royo, R., & Guijarro, F. (2020). Forecasting stock market trend: A comparison of machine learning algorithms. *Finance, Markets and Valuation*, 6, 37–49.
- Chung, F. R., & Graham, F. C. (1997). *Spectral graph theory*. 92. American Mathematical Soc.
- Cornuejols, G., & Tütüncü, R. (2006). *Optimization methods in finance* volume 5. Cambridge University Press.
- Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems* (pp. 3844–3852).
- Drożdż, S., Grümmer, F., Ruf, F., & Speth, J. (2001). Towards identifying the world stock market cross-correlations: Dax versus dow jones. *Physica A: Statistical Mechanics and its Applications*, 294, 226–234.
- Elton, E. J., Gruber, M. J., Brown, S. J., & Goetzmann, W. N. (2009). *Modern portfolio theory and investment analysis*. John Wiley & Sons.
- Fama, E. F. (1965). The behavior of stock-market prices. *The journal of Business*, 38, 34–105.
- Farmer, R. E. (2012). The stock market crash of 2008 caused the great recession: Theory and evidence. *Journal of Economic Dynamics and Control*, 36, 693–707.
- Felmer, P., Quaas, A., & Tan, J. (2012). Positive solutions of the nonlinear schrödinger equation with the fractional laplacian. *Proceedings of the Royal Society of Edinburgh Section A: Mathematics*, 142, 1237–1262.
- Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In *Brazilian symposium on artificial intelligence* (pp. 286–295). Springer.
- Gama, J., Sebastião, R., & Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine learning*, 90, 317–346.
- Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46, 44.
- García, F., González-Bueno, J., Oliver, J., & Riley, N. (2019a). Selecting socially responsible portfolios: a fuzzy multicriteria approach. *Sustainability*, 11, 2496.
- García, F., González-Bueno, J., Oliver, J., & Tamošiūnienė, R. (2019b). A credibilistic mean-semivariance-per portfolio selection model for latin america. *Journal of Business Economics and Management*, 20, 225–243.
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 1263–1272). JMLR. org.
- Grondman, I., Busoniu, L., Lopes, G. A., & Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42, 1291–1307.
- Hammond, D. K., Vandergheynst, P., & Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30, 129–150.
- Henaff, M., Bruna, J., & LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, .

- Hu, Y., Liu, K., Zhang, X., Xie, K., Chen, W., Zeng, Y., & Liu, M. (2015). Concept drift mining of portfolio selection factors in stock market. *Electronic Commerce Research and Applications*, *14*, 444–455.
- Hussain, A. J., Knowles, A., Lisboa, P. J., & El-Deredy, W. (2008). Financial time series prediction using polynomial pipelined neural networks. *Expert Systems with Applications*, *35*, 1186–1199.
- Jiang, Z., Xu, D., & Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*, .
- Kingma, D., & Ba, J. (2015). Adam: a method for stochastic optimization (2014). *arXiv preprint arXiv:1412.6980*, *15*.
- Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, .
- Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems* (pp. 1008–1014).
- Langdon, D. S., McMenamin, T. M., & Krolik, T. J. (2002). Us labor market in 2001: Economy enters a recession. *Monthly Lab. Rev.*, *125*, 3.
- Langr, J., & Bok, V. (2019). *GANs in Action: Deep Learning with Generative Adversarial Networks*. Manning Publications.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, .
- Lucey, B. M., & Muckley, C. (2011). Robust global stock market interdependencies. *International Review of Financial Analysis*, *20*, 215–224.
- Magdon-Ismail, M., & Atiya, A. F. (2004). Maximum drawdown. *Risk Magazine*, *17*, 99–102.
- Markowitz, H. M. (1978). Portfolio selection. *Wiley, New York*, .
- Maverick, J. (2019). What is a good sharpe ratio? <https://www.investopedia.com/ask/answers/010815/what-good-sharpe-ratio/>
- Meng, Q., Catchpoole, D., Skillicom, D., & Kennedy, P. J. (2017). Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 364–371). IEEE.
- Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin.
- Nti, I. K., Adekoya, A. F., & Weyori, B. A. (2019). A systematic review of fundamental and technical analysis of stock market predictions. *Artificial Intelligence Review*, (pp. 1–51).
- Omidi, F., Abbasi, B., & Nazemi, A. (2017). An efficient dynamic model for solving a portfolio selection with uncertain chance constraint models. *Journal of Computational and Applied Mathematics*, *319*, 43–55.
- Ormos, M., & Urbán, A. (2013). Performance analysis of log-optimal portfolio strategies with transaction costs. *Quantitative Finance*, *13*, 1587–1597.
- Park, K., & Shin, H. (2013). Stock price prediction based on a complex interrelation network of economic factors. *Engineering Applications of Artificial Intelligence*, *26*, 1550–1561.
- Pouya, A. R., Solimanpur, M., & Rezaee, M. J. (2016). Solving multi-objective portfolio optimization problem using invasive weed optimization. *Swarm and Evolutionary Computation*, *28*, 42–57.
- Qiu, M., & Song, Y. (2016). Predicting the direction of stock market index movement using an optimized artificial neural network model. *PloS one*, *11*, e0155133.
- Rocchi, J., Tsui, E. Y. L., & Saad, D. (2017). Emerging interdependence between stock values during financial crashes. *PloS one*, *12*.
- Rockafellar, R. T., Uryasev, S. et al. (2000). Optimization of conditional value-at-risk. *Journal of risk*, *2*, 21–42.

- Sarwar, G., & Khan, W. (2019). Interrelations of us market fears and emerging markets returns: Global evidence. *International Journal of Finance & Economics*, *24*, 527–539.
- Serletis, A., & Rosenberg, A. A. (2009). Mean reversion in the us stock market. *Chaos, Solitons & Fractals*, *40*, 2007–2015.
- Sharpe, W. F. (1994). The sharpe ratio. *Journal of portfolio management*, *21*, 49–58.
- Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., & Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, *30*, 83–98.
- Shuman, D. I., Ricaud, B., & Vandergheynst, P. (2016). Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, *40*, 260–291.
- Shuman, D. I., Vandergheynst, P., & Frossard, P. (2011). Chebyshev polynomial approximation for distributed signal processing. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)* (pp. 1–8). IEEE.
- Soleymani, F., & Paquet, E. (2020). Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder-deepbreath. *Expert Systems with Applications*, (p. 113456).
- Stone, A., Wang, H., Stark, M., Liu, Y., Scott Phoenix, D., & George, D. (2017). Teaching compositionality to cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5058–5067).
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, *3*, 9–44.
- Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems* (pp. 1057–1063).
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*, .
- Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*, 279–292.
- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, .
- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, .
- Yue, W., & Wang, Y. (2017). A new fuzzy multi-objective higher order moment portfolio selection model for diversified portfolios. *Physica A: Statistical Mechanics and its Applications*, *465*, 124–140.
- Zhang, D., Hu, M., & Ji, Q. (2020). Financial markets under the global pandemic of covid-19. *Finance Research Letters*, (p. 101528).
- Zhang, S., Tong, H., Xu, J., & Maciejewski, R. (2019). Graph convolutional networks: a comprehensive review. *Computational Social Networks*, *6*, 11.
- Zhao, D., Wang, H., Shao, K., & Zhu, Y. (2016). Deep reinforcement learning with experience replay based on sarsa. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1–6). IEEE.