# Neural stochastic control application: Optimal portfolio allocation

**Kevin Noel l**[1] **, Michal Fabinger** [2]
[1]ING, Rakuten [2]T.D.S.

## Abstract

Neural Control (NC) is a growing field in stochastic optimal control applied to various dynamical systems such as quadcopter. NC is a non-parametric, learning based computational scheme, capable of handling high dimensional control problems. In this paper, "NC" principles have been applied to the problem of optimal investment portfolio allocation, which is commonly solved by the usage of covariance matrix in a Markowitz framework. Therefore, the portfolio quadratic optimization problem has been first re-cast into a generic stochastic control framework with non-linear dependencies and then solved using various Neural Networks (NNs). In NC context, literature on stacked Feed-Forward for time independent control has been recalled to formulate the properties of new NN architectures: Long Short-Term Memory (LSTM) and Attention LSTM for time-dependent control. Final section presents the numerical results of control weights convergence and dynamics using Monte Carlo simulation of portfolio asset prices (Geometric Brownian Motion) through various scenarios: first using stationary regime of volatility and correlation, second using non-stationary regime. Thus, behaviour of three NN architectures has been studied in those scenarios. Results suggest the further development of NC in domain of portfolio allocation as alternative of covariance matrix based solutions and related finance, economic supply and demand problems.

## Introduction

Stochastic control problems are usually solved through the principle of dynamic programming or Monte Carlo methods such as Longstaff and Schwartz methods (Longstaff and Schwartz, 2001; Bertsimas and Lo, 1998; Salas and B, 2013). However, these mathematical methods usually confront technical difficulties when dealing with high dimensional search space problems, which commonly refers to the "curse of dimensionality" (Powell, 2011). Recent advances in Artificial Intelligence (AI) have shown the efficiency of a trained NN to efficiently solve non-linear high dimensional search space problems (i.e. Alpha Zero type algorithms (Laterre et al., 2018) in polynomial time. Thus, NC leverages the computational efficiency of NNs for stochastic optimal control problems.



NC has recently gained attraction on solving problems for dynamical systems. We can quote quad-copter controller as an example of a neural controller used in practical schemes (Izzo, Tailor, and Vasileiou, 2018; Shi et al., 2018) where NN has been used as an adjuster for the residual error of the motion equations. Authors provide various procedures to ensure the stability of the NN control output.Shi et al. (2018) present Lipchitz-based weight normalization to obtain guarantees on the trajectories, while Izzo, Tailor, and Vasileiou (2018) present an advanced Taylor scheme decomposition to assess the robustness of such neural controlled trajectories. Han and others (2016) show the usage of NN as an approximate solver for high dimensional Backward Stochastic Differential Equations (BSDE) using a Feed-Forward architecture. Among those BSDE, optimal control equation, Hamilton Jacobi Bellman (HJB), used in portfolio allocation optimization, has been evaluated successfully.

We note that reinforcement learning deals mostly with infinite time horizon period and model free states, whereas optimal control deals with finite time horizon and parametric states (i.e. states based on model equation). Although, both fields have characteristics in their approach, NC tends to encompass both fields together as the focus shifts towards the computing scheme design (i.e. NN architecture) and learning process beyond a parametric and calibration approach(i.e. formulae structural model). In addition, "reinforcement learning" can also be called as an alternative way to train a neural controller (besides supervised learning scheme for optimal state-action) and optimal control is sometimes called (improperly) "model-based reinforcement learning" in the machine learning community.

Neural controllers rely on the universal approximator property of NNs to use them as proxy for a complex and non-linear control process. NN learns the approximation through either Monte Carlo simulation or physical measured data. The approach of "neural controller" represents a paradigm shift where instead of having parametric formulae, a generic learning based, non explicit algorithmic approach is used. However, for critical systems development, such computing process requires the need of having mathematical guarantees of the NN output (Izzo, Tailor, and Vasileiou, 2018; Shi et al., 2018).

This paper proposes to extend the usage of "NC" in the field of optimal portfolio allocation. Conventionally, portfolio allocation is a well-studied example of optimal control or optimization problem Perrin and Roncalli (2019) which lies between HJB formulation (Constant Relative Risk Aversion (CRRA) based) and Markowitz (mean variance utility) framework Czichowsky (2013); Wang (2013); Michaud (2008). In addition, the risk budgeting framework, still relying on covariance matrix estimation, has been added recently (Richard and Roncalli, 2019; Perrin and Roncalli, 2019). Extensive treatment of optimal portfolio allocation has been developed over the years by the quantitative finance community. However, most of the schemes rely on refined estimation of the asset return covariance matrix. NC compute scheme is considered the future for portfolio allocation due to its capacity of handling non-linear dependencies and very high dimensional search space. This paper characterizes first results of neural controlled portfolio allocation based on specific architectures versus the baseline (covariance matrix based allocation).

## Stochastic Control Formulation

To cast the portfolio allocation problem, generic discrete stochastic control framework is being recalled Cme Hur (2019); Huyen Pham (2019); Henry-Labordere (2017). The dynamic of the controlled state process $\boldsymbol{X} = (X_n)$, valued in $\mathbb{R}^d$, is described by the state equation given as follows

$$X_{n+1} = F\left(X_n, a_n, W_{n+1}\right), \tag{1}$$

where

$$\begin{aligned} X_n &= (X_{1,n}, \ldots, X_{d,n}) \\ a_n &= (a_{1,n}, \ldots, a_{d,n}) \end{aligned} \tag{2}$$

with the time frame ($t_0 = 0, \ldots, t_N = T$). The process $(W_n)$ is defined on some probability space $(\Omega, \mathcal{F}, P)$ equipped with the filtration $H = (H_n)$, generated by the noise $(W_n)$. The control $\boldsymbol{a} = (a_n)$ is a process valued in $A \subset R^q$, and $F$ is a measurable function from $R^d \times R^q \times E$ into $R^d$, typically the Euler scheme for a controlled diffusion process. Therefore, the distribution of $(W_n)$ and the choice of control $\boldsymbol{a}$ should determine the behavior of the process completely.

Given a running cost function $f$ defined on $R^d \times R^q$, and a terminal cost function $g$ defined on $R^d$, cost function $J(\boldsymbol{a})$ associated with a control process $\boldsymbol{a} = (a_n)$ can be expressed as follows:

$$J(\boldsymbol{a}) = \mathbb{E}\left[\sum_{n=0}^{N} f\left(X_n, a_n\right) + g\left(X_N\right)\right] \tag{3}$$

The control problem considered is a Markov Decision Process (MDP), which can be described through the equation 4.

$$V_0(X = x_0) = \inf_{\boldsymbol{a}} J(\boldsymbol{a}) \tag{4}$$

The value function $V$ is the discrete-time approximation for the solution of a fully non-linear HJB equation. The goal is to find an optimal control $\boldsymbol{a}^* \in A^N$, i.e., which attains the optimal value for time $n = 0$:

$$V_0\left(x_0\right) = J\left(\boldsymbol{a}^*\right). \tag{5}$$

This global dynamic optimization problem (4) can be reduced to local optimization sub-problems via a Dynamic Programming (DP) approach (linear quadratic case), which allows to determine the value function in a backward recursion as in Equation 6 and Equation 7.

Terminal conditions:

$$V_n(x) = g(x),\ \forall x \in \mathbb{R}^d. \tag{6}$$

Backward recurrent formulae: going from $n = N - 1$ to 0, and $(x, a) \in \mathbb{R}^d \times A$

$$\begin{aligned} \mathrm{V_n}(x) &= \inf_{a \in A} Q_n(x, a) \\ Q_n(x, a) &= f(x, a) + \mathbb{E}\left[V_{n+1}\left(X_{n+1}\right) | X_n = x, A_n = a\right] \end{aligned} \tag{7}$$

The function $Q_n$ is called the optimal state-action value function, and $V_n$ is the (optimal) value function.

The practical implementation of the DP formula may suffer from the curse of dimensionality and complexity when the state space dimension $d$ and the control space dimension $q$ are high. NC can be an alternative solution for such high dimensional or non-linear problem. Depending on the nature of the problem, different NN architectures can be used, transferring the equation based parametric problem into a problem of searching an efficient and generic neural computation scheme. There are significant progresses to be made into identifying the correct neural compute schemes for specific parametric control problems.

Here, the portfolio allocation problem has been cast into this control framework with the classical mean variance cost function:

$$J(\boldsymbol{a}) = -\gamma\, \mathbb{E}\left[R_N(\boldsymbol{X}, \boldsymbol{a})\right] + \frac{1}{2} \operatorname{Var}\left[R_N(\boldsymbol{X}, \boldsymbol{a})\right] \tag{8}$$

$$\begin{aligned} R_n\left(\boldsymbol{x}, \boldsymbol{a}\right) &= \sum_{k=1}^{d} a_{k,n-1} \cdot r_{k,n} \\ r_{k,n} &= \log\left[\frac{x_{k,n}}{x_{k,n-1}}\right] \end{aligned} \tag{9}$$

with

$$X_{n+1} = F\left(X_n, a_n, W_{n+1}\right) \text{ and } \sum_{i=1}^{d} a_{i,n} = 1$$

- $(X_n)$ is the state vector of the price process at time tn.
- $(R_n)$ , $(r_{k,n})$ are the log-returns of the portfolio and the d assets.
- $F$ is the function describing the path dynamic of $(X_n)$.
- $(an)$ is the control vector (value control).
- $\gamma$ is the risk preference factor (constant in this problem).

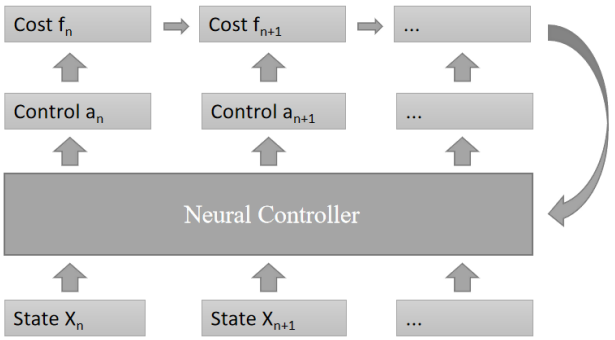



Figure 1: Compute scheme for NC

In case that $(X_n)$ follows a multivariate Geometric Brownian motion, with respect to historical probability measure, we have :

$$x_{k,n} = x_{k,n-1}.exp[(\mu_k - 1/2.\sigma_k^2)\Delta t_n + \sigma_k.W_{k,n}]$$
$$< W_{i,n}, W_{j,n} >= \Omega_n^{i,j}.\Delta t_n \tag{10}$$

$\mu$, $\sigma$, $W_{k,n}$ , $\Omega$ being the vector drift, vector volatility,the series of correlated Brownians and the correlation matrix of the Brownians.

Here, the so called "pre-committed" strategy of the mean variance optimal control problem is being used, which is "Bellman" optimal at $t_0 = 0$ only. See (Czichowsky, 2013) for extensive treatment of this part as well as a game theory approach of optimality (i.e. Pareto and Nash optimality).

## Neural Network Formulation

This section presents the approximation of the control $(a_n)$ and its dependence on $X_n$ through various NN architectures. $a_n(X_n)$ has been represented with the approximation presented in Equation 11:

$$a_n\left(\mathrm{X}_n\right) \approx a_n\left(\mathrm{X}_n, \theta\right) \tag{11}$$

where $\theta$ are the parameters of the NN (globally shared across all the time steps $0 \ldots N$). Justification of this parametrization is the fact that NNs are rigorously proven universal approximators Goodfellow, Bengio, and Courville (2016).

The computational scheme is illustrated in Figure 1.

Previous works in the field have been based on using FNNs and different learning processes for the total cost (value and policy based) Izzo, Tailor, and Vasileiou (2018); Han and others (2016); Shi et al. (2018).

As contributions of this paper, the portfolio allocation problem has been reformulated from a quadratic optimization problem into stochastic control learning problem. Then, to solve this learning problem, various NN architectures have been. employed: Feed-Forward Network (FNN), a LSTM NN which handles time-dependent hidden states, and Attention LSTM NN where specific weights are able to focus on specific parts of the input state $(X_n)$. Both LSTM and Attention LSTM are novel architectures used for the optimal control problem. Further, the effect of NN architectures versus pure optimization baseline for this learning problem has been characterized. One reason of using NN is the presence of non-linear temporal residual dependencies between the states (none taken into account in the equations). In the case of portfolio allocation, those dependencies are usually translated into hard allocation constraints (as prior information) to ensure portfolio stability.

### Algorithm

**Algorithm 1** Neural Controller

**Require:**
$x_0, \sigma$ : Initial state and asset dynamics
$N$ : Final index for Time horizon
$K$ : Number of epochs
$I$ : Batch size
**Parameters:** $\theta$ {NN Parameters}
**Initialize parameters**
**Initialize states**
**for** $k = 1$ **to** $K$ **do**
  **for** $i = 1$ **to** $I$ **do**
    **for** $n = 1$ **to** $N$ **do**
      Sample correlated Brownians:
      $W \sim N(0, 1)$
      Update system state:
      $X^{(i)}(n) = F(X^{(i)}(n-1), A^{(i)}(n-1), W)$
      Estimate control value:
      $A^{(i)}(n) = f_{\mathrm{NN}}(X^{(i)}(n), \theta)$
    **end for**
  **end for**
  Compute Mini Batch loss:
  $\mathrm{Loss} = \frac{1}{I} \sum_{i=1}^{I} J(A^{(i)}, X^{(i)}) + \mathrm{Reg}(\theta)$
  Update parameters by gradient descent:
  $\theta \leftarrow \theta - Adam\left(\frac{\partial \mathrm{Loss}}{\partial \theta}\right)$
**end for**
**return** $\theta(K)$

Algorithm 1 essentially learns a value-based policy for the control, which can be done using different choices of NN architectures. The regularization part (Reg) used here is L1 (LASSO), which enforces sparsity in the NN weights. NNs are usually trained through gradient-based optimizer and in this research, Adam, Kingma (2015), has been used as well as exponential smoothing average for the weights (removing jittering effect). Implementation was done using Tensorflow 1.14, Abadi et al. (2015).

### Feed-Forward Neural Network

This section presents the approximation of the optimal control through a series of FNNs stacked together over the time

steps $0 \dots N$. For clarity, Han and others (2016) architecture for BSDE problems has been followed in this research to describe the architecture of FNNs.

Each FNN at each time step is obtained by successive compositions of one-dimensional non-linear functions, stacked into layers in a chain structure, with each layer being a function of the layer that preceded it. The FNNs do not share the weights across them and approximate the control function independently at each time step.

Given $M$ one-dimensional non-linear functions composed successively with $\tanh$ as the non-linear activation function for the hidden layers, the sigmoid ($\sigma$) for the output layer, and the NN is described by $a_{n,M}$ with input variable $x_n$, then the network can be described as following:

$$y = a_{n,M}(x_n) \tag{12}$$

with

$$a_{n,0} = x_n \in [0, 1], \tag{13}$$

for $l = 1, 2, \dots, M-1$

$$a_{n,l} = \tanh\left(W_{n,l} a_{n,l-1} + b_{n,l}\right) \tag{14}$$

and finally for the last layer

$$a_{n,M} = \sigma\left(W_{n,M} a_{n,M-1} + b_{n,M}\right) \in [0, 1] \tag{15}$$

Where, $M$ represents the total number of layers, $M-1$ denotes the number of hidden layers, $W$ denotes the weights (learnable parameters) and $b$ denotes the biases which can be combined together to form $\theta$.

Although approximation performance has been shown in Han and others (2016), given the architecture, Stacked FNNs can only be used for finite time control problems.

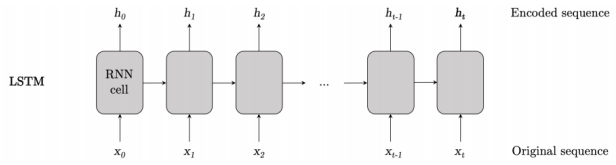
## Long Short-Term Memory (LSTM)

Sequential data processed through Feed-Forward network usually possess separate parameters for each time step input. Consequently, fails to generalize to sequence lengths unseen during training, and is unable to share statistical strength across different positions in time. LSTMs (and RNNs) address this problem by embracing parameter sharing through the hidden states. When compared to a FF with sub-network, an LSTM network reuses previous hidden states from a previous time step to predict the values for the next time step, learning the full period series pattern.

LSTMs deploy a gating mechanism that controls the flow of information and helps to capture the long-term temporal dependencies, which is missed in vanilla RNNs where the gradients propagated over many stages tend to either vanish or explode. An LSTM cell has a self-loop, as well as the outer recurrence of the RNN. Each cell not only has the same inputs and outputs as an ordinary recurrent network but also more parameters and a system of gating units.

**Unrolled LSTM architecture:** At each time step, the input state $x_n$ and the previous hidden state $h_{n-1}$ are provided to the LSTM cell. The output of the LSTM cell at each time step is the control vector $a_n$.

A standard LSTM cell has three gates: an update gate $u_n$ (input gate), a forget gate $f_n$, and an output gate $o_n$. The




LSTM also maintains a cell state $c_n$ at time $n$ along with receiving the previous cell state $c_{n-1}$.

The weights for an LSTM layer are listed as follows:

- Input weights: $W^z, W^u, W^f, W^o$
- Recurrent weights: $R^z, R^u, R^f, R^o$
- Bias weights: $b^z, b^u, b^f, b^o$

If $x_n$ is the input vector at time $n$, and $h_{n-1}$ is the previous cell activation, then forward pass can be written as follow: Gates updates

$$\hat{u}_n = W^u x_n + R^u h_{n-1} + b^u \qquad u_n = \sigma\left(\hat{u}_n\right) \tag{16}$$

$$\hat{f}_n = W^f x_n + R^f h_{n-1} + b^f \qquad f_n = \sigma\left(\hat{f}_n\right) \tag{17}$$

$$\hat{o}_n = W^o x_n + R^o h_{n-1} + b^o \qquad o_n = \sigma\left(\hat{o}_n\right) \tag{18}$$

Cell state updates

$$\hat{z}_n = W_z x_n + R^z h_{n-1} + b^z \qquad z_n = g\left(\hat{z}_n\right) \tag{19}$$

$$c_n = z_n \odot u_n + c_{n-1} \odot f_n \tag{20}$$

Lastly, the hidden unit is activated as follows:

$$h_n = p\left(c_n\right) \odot o_n \tag{21}$$

Where $\sigma$(logistic sigmoid) and $g(x) = p(x) = \tanh(x)$ represent point-wise non-linear activation functions, and $\odot$ denotes the Hadamard product.

## Attention LSTM

In vanilla LSTM, when tasks involving sequential data (e.g. time series) are performed, the LSTM model encodes a whole input sequence into a fixed-length context vector and focuses on the entire input sequence to assign the probability of data using a Markov assumption. However, nn Attention LSTM, Bahdanau, Cho, and Bengio (2014), the model focuses only on part of a subset of information relevant to the generation of the next target output and frees the model from encoding the full fixed-length context vector. Having the functionality of focusing on the part of sequence, Attention LSTM also helps in better identifying specific part of the sequence relevant for the prediction or estimation. Several attention mechanisms have been developed over recent years such as Content-base, Additive, Location-based and Dot-Product.

In general case, considering a plain LSTM with an input $\boldsymbol{x} = (x_1, \dots, x_N)$, a hidden state $(h_n)$, and an output $(y_n)$, it can be decomposed into an encoder part (encoder LSTM),

and a decoder part (LSTM decoder). If a non-linear function $q$, and $c$ an encoding vector generated from the sequence of the hidden states $(h_n)$ are added, then LSTM encoder can be expressed as follows:

$$h_n = \text{EncoderLSTM}\left(x_n, h_{n-1}\right) \tag{22}$$

$$c = q\left(h_1, \ldots h_{n-1}\right) \tag{23}$$

The decoder part of the LSTM then predicts the output $y_n$ at time step $n$ given the full context vector $c$ and all the previously predicted values $\{y_1, \ldots, y_{n-1}\}$. Then, the probability of the output vector $y$ is expressed by decomposing the joint probability into the ordered conditional:

$$p(y) = \prod_{n=1}^{N} p\left(y_n | y_1, \ldots, y_{n-1}, c\right) \tag{24}$$

In this case, the output of the LSTM cell (plain or attention) is the control vector $a_n = y_n$.

In Attention LSTM, this probability is conditioned on a distinct context vector $c_i$ for each target input $y_i$. If $h_i$ is an LSTM hidden state, and $g$ is a non-linear function, probability can be expressed as follows:

$$p\left(y_i | y_1, \ldots, y_{i-1}, c\right) = g\left(y_{i-1}, h_i, c_i\right) \tag{25}$$

If an Attention LSTM encoder maps the input sequence $(x_1, \ldots, x_n)$ to $(h_1, \ldots, h_n)$, then the context vector $c_i$ is given by:

$$c_i = \sum_{j=1}^{n} \mu_{i,j} h_j \tag{26}$$

Here, $\mu_{i,j}$ denotes the "amount of attention" $y_{i-1}$ should give to $h_j$ in order to predict $y_i$. Context attention weights $\mu_{i,j}$ are given by:

$$\mu_{i,j} = \frac{\exp\left(e_{i,j}\right)}{\sum_{k=1}^{n} \exp\left(e_{i,k}\right)} \tag{27}$$

Where $e_{i,j} = \text{score}\left(y_{i-1}, h_j\right)$ is an alignment model, which scores how the input at position $j$ matches output at position $i$ (playing the role of "correlation" score between input/ouput). Several alignment models are available, such as vector dot product, $\cos$ or $\tanh$, see Bahdanau, Cho, and Bengio (2014) for the details. For this research,$\tanh$ scoring expressed by following equation has been used:

$$e_{i,j} = V_a . \tanh(W_a . y_{i-1} + U_a . h_j) \tag{28}$$

where $V_a, W_a, U_a$ represent the attention matrices.

The Attention mechanism has several variations including "soft attention" and "hard attention". In the case of "soft attention", the weights are learned and placed "softly" all over the input source which makes the model differentiable and smooth at the expense of computational cost (algorithm complexity is quadratic). However, in the case of "hard attention", one part of the input is selected at a time, which lowers the complexity cost but makes the model non-differentiable (additional training methods are required).

## Results with Optimal Portfolio allocation

Application of optimal control in quantitative finance area is mostly related to portfolio allocation with the application of HJB. However, from practitioner side, the seminal work of Markowitz Markowitz (1952) with his Mean Variance (MV) optimal portfolio has widely adopted along with quadratic optimization schemes Perrin and Roncalli (2019). MV can still be cast into an optimal control framework through the usage of mean variance utility into the cost function (equation (8)). Major advantage of MV is its capability of simply modelling the investor divergent objectives (return, risk).

Although MV has been criticized over the years for its strong sensitivity to input estimated parameters, it is still the most widely used optimal allocation framework and baseline model. Recent works lead to new portfolio allocation framework such as Equally Risk Contribution (ERC) Richard and Roncalli (2019), which relies on Euler decomposition of variance for risk-based allocation.

However, those allocation frameworks such as MV, ERC relies on the estimation of the covariance matrix. It has been shown in Malevergn and Sornette (2006) that asset interactions are non-linear and exhibits long-range memory which makes covariance matrix based frameworks limited by the parametric forms less optimal to deal with general non-linear interactions.

In this section, MV has been used as baseline results of the optimal weights and shown how neural controller is able to learn from the dynamics of assets and their interactions, the optimal weights under various correlation scenarios and thus, enabling a framework for optimal portfolio allocation beyond covariance matrix estimation paradigm.

### Baseline review

Recall the genetic form of the mean variance cost function expressed by the equation as follows :

$$J(\boldsymbol{a}) = -\gamma\, \mathbb{E}\left[R_N(\boldsymbol{X}, \boldsymbol{a})\right] + \frac{1}{2} \operatorname{Var}\left[R_N(\boldsymbol{X}, \boldsymbol{a})\right] \tag{29}$$

**Mean Variance Portfolio** For the baseline control weights, following cost function has been minimized over a past period $[t_{i-1}, t_i]$, with d assets:

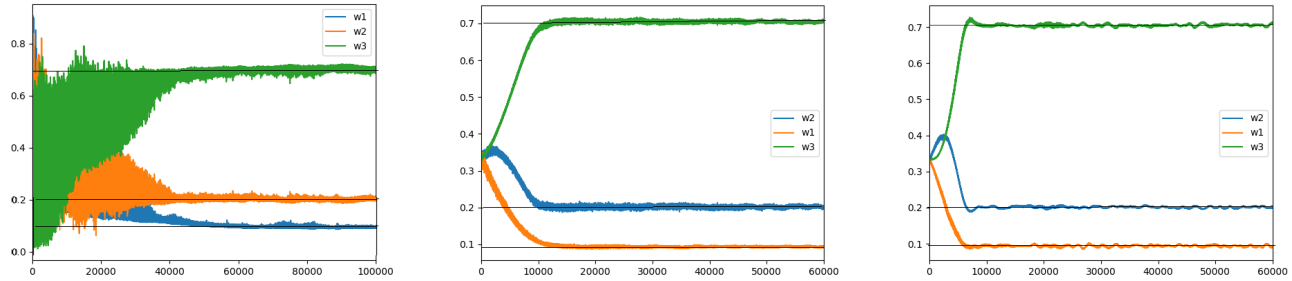
$$J_i\left(a_i^-\right) = -\gamma R_i^T \cdot a_i^- + \frac{1}{2} a_i^- t \cdot \Sigma_i \cdot a_i^- + Reg\left(a_i^-\right) \tag{30}$$

with $a_i^- = (a_{k,i})$, $a_{k,i} > 0$, $\sum_{k=1}^{d} a_{k,i} = 1$ (no friction, no short-sell conditions)
$\Sigma_i$ : Estimated covariance matrix of the d components.
$R_i$ : Estimated return vector of the d components
$Reg\left(a_i\right)$ : Penalty or regularization term.

Using Lagrange multiplier, the solution is given by Glasserman and Xu (2013); Baviera and Bianchi (2019):

$$\begin{aligned} A = \mathbf{1}^T \Sigma^{-1} \boldsymbol{\mu} \qquad C = \mathbf{1}^T \Sigma^{-1} \mathbf{1} \\ \mathbf{a}_i^* = \gamma \frac{\Sigma^{-1} \boldsymbol{Ri}}{A} + (1 - A\gamma) \frac{\Sigma^{-1} \mathbf{1}}{C} \end{aligned} \tag{31}$$

$\mathbf{1}$ : Unit vector of d components




(a) Feed-Forward NN (b) LSTM NN (c) Attention LSTM NN

Figure 2: Stationary regime weights for different NN, at tn=T

Optimal weights are then expressed as a function of the inverse of covariance matrix (i.e. Precision Matrix).

**Stationary regime** In this section, it has been considered that each of the $d$ components of $(X_n)$ follow correlated Geometric Brownian Motions (GBM), and expressed in the discrete case (using historical measure):

$$\begin{aligned} \Delta R_{n,k} &= (m_{k,n}) \cdot \Delta t_n + \sigma_{k,n}.\Delta W_{k,n} \\ \Delta t_n &= t_n - t_{n-1}, t_n = 0, t_N = T, k = 1 \dots d \\ < \Delta W_n^i, \Delta W_n^j > &= \rho_n^{i,j} \Delta t_n \end{aligned} \quad (32)$$

$R_{k,n}, m_{k,n}, \sigma_{k,n}$ are respectively the component price returns, drift, and volatility. For Monte Carlo simulation details, standard Euler scheme Glasserman and Xu (2013) has been used.

In stationary regime, correlation is considered as fixed over the full period and following parameters are set:

- $N = 60$ , $T = 3$ year, $k = 1 \dots 3$ , $d = 3$ assets, $\gamma = 0$
- $m_{k,n} = 10\%$ $\sigma_{k,n}(\text{k} = 1 \dots 3) = 10\%^*\text{k}$
- $\rho_n^{i,j} = 0$ (constant)

Parameters of the NN training are:

- M=10000, LR= 0.002, Mini-batch=64, LASSO reg.
- FF layers : 4, Layer dimensions : (3,10,10,3)
- LSTM hidden dim : 5, Attn. LSTM hidden dim : 5

For each of the NC, the convergence of control weights $(a_n)$ at $t_n = 3$ years has been plotted. Constant lines in the plot represent the baseline MV weights.

It has been observed from the results that LSTM NN, and Attention LSTM NN converges faster in comparison to FF NN, converged respectively at 10k, 8k, 50k. Also, Attention NN was found to has the most stable convergence (i.e. Std. Dev of the convergence curve is smallest).

## No stationary regime

In this section correlation and volatility are time varying which simulate a regime change behavior. From equation (32), the current dynamic framework is being added:

$$\begin{aligned} &0 < n < T1 : \\ &\qquad \sigma_n^{i,j} = (0.21, 0.20, 0.095) \\ &\qquad \rho_n^k = (-0.05, 0.15, -0.13) \\ &T_1 \leq n < T_2 : \\ &\qquad \sigma_n^{i,j} = (0.12, 0.115, 0.225) \\ &\qquad \rho_n^k = (-0.05, 0.15, -0.13) \\ &T_2 \leq n < T_3 : \\ &\qquad \sigma_n^{i,j} = (0.30, 0.23, 0.14) \\ &\qquad \rho_n^k = (-0.05, 0.15, -0.13) \end{aligned} \quad (33)$$

- $d = 3$ assets, $N = 30$ , $Tk = (10, 22, 30)$ steps, $k = 1 \dots 3$ , $\gamma = 0$, $m_{k,n} = 0.01$
- $\rho_n^{i,j}$ : Upper diagonal (i,j) asset pairs, $i, j = 1 \dots 3$

Parameters of the NN training are:

- M=100000, LR= 0.002, Mini-batch=16, L1 Reg.
- FF layers : 4, Layers dim. : (3,15,15,3)
- LSTM hidden dim : 30, Output layers dim : (15,3)
- Attn. LSTM hidden dim : 30, Output layers dim : (15,3)

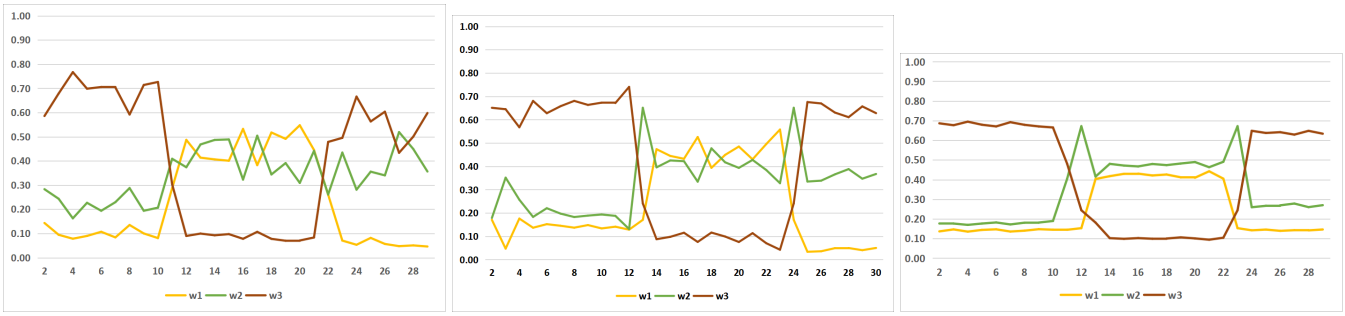
**Simulation results:** Following is the table of convergence error for the control weights (vs baseline exact formulae ), expressed as MAE at path p :

$$MAE(p) = \frac{1}{N} \sum_{k=1..3}^{n=1...30} \left| a_{k,n,p}^{NN} - a_{k,n}^{exact} \right| \quad (34)$$

Table 1: Mean absolute error of the control weights

| Path | FF | LSTM | Attn LSTM |
|---|---|---|---|
| 100 | 25.5% | 18.7% | 11.0% |
| 1000 | 23.7% | 22.5% | 14.3% |
| 5000 | 23.0% | 21.9% | 12.4% |
| 10000 | 20.6% | 22.2% | 10.6% |
| 50000 | 9.4% | 8.2% | 6.5% |
| 75000 | 3.6% | 4.0% | 3.6% |
| 100000 | 4.5% | 4.3% | 3.4% |

All 3 NC are able to learn the regime change in volatility levels, however, Attention LSTM NC having able to




(a) Feed-Forward NN (b) LSTM NN (c) Attention LSTM NN

Figure 3: ample of control weights over the 3 years time period, at path=20000.

learn optimal weights faster than other NNs. Weights of FFs NC being clearly more volatile than 2 others as the convergence table shows. In addition to previous studies which focused mainly on terminal value convergence Han and others (2016), full path control weights convergence is being as important in a control problem.

## Conclusion

In this paper, problem of stochastic optimal control was recalled for its formalization and recent applications of NC for dynamical systems. Furthermore, problem of optimal portfolio allocation with finite horizon was cast into this framework. Research proposed neural network approach as a potential solution to address high dimensional control problem, called "Neural Control". In the case of portfolio optimization, paper highlighted the paradigm change from covariance matrix in how to solve it. Approximation of control using NNs was demonstrated through the implementation of different neural architectures: Feed-Forward network, LSTM and Attention LSTM. From the results section, it has been observed that NN are able to learn optimal portfolio weights through a learning process (instead of QP) and have characterized the various architecture behaviors. In future work, we expect to investigate further new NN architecture as well as stability process fo the control

## References


Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; and Chen, Z. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural Machine Translation by Jointly Learning to Align and Translate.

Baviera, R., and Bianchi, G. 2019. Model risk in mean-variance portfolio selection: an analytic solution to the worst-case approach. Technical report.

Bertsimas, D., and Lo, A. W. 1998. Optimal control of execution costs. *Journal of Financial Markets, 1(1):150*.

Czichowsky, C. 2013. Time-consistent mean-variance portfolio selection in discrete and continuous time. *Finance and Stochastics* 17(2):227–271.

Cme Hur, Huyłn Pham, X. W. 2019. Some machine learning schemes for high-dimensional nonlinear pdes.

Glasserman, P., and Xu, X. 2013. Robust risk measurement and model risk. *Quantitative Finance*.

Glasserman, P. 2003. *Monte Carlo Methods in Financial Engineering*, volume 53 of *Stochastic Modelling and Applied Probability*. New York, NY: Springer New York.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep learning*. MIT press.

Han, J., et al. 2016. Deep learning approximation for stochastic control problems. *NIPS 2016, Deep Reinforcement Learning Workshop*.

Henry-Labordere, P. 2017. Deep primal-dual algorithm for bsdes: Applications of machine learning to cva and im. *Available at SSRN: https://ssrn.com/abstract=3071506*.

Huyen Pham, Achref Bachouch, N. L. C. H. 2019. Deep neural networks algorithms for stochastic control problems on finite horizon.

Izzo, D.; Tailor, D.; and Vasileiou, T. 2018. On the stability analysis of optimal state feedbacks as represented by deep neural models. *arXiv preprint arXiv:1812.02532*.

Kingma, D.P., B. J. 2015. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations, San Diego* arXiv:1412.6980.

Lai, T. L.; Xing, H.; and Chen, Z. 2011. Mean-variance portfolio optimization when means and covariances are unknown.

Laterre, A.; Fu, Y.; Jabri, M. K.; Cohen, A.-S.; Kas, D.; Hajjar, K.; Dahl, T. S.; Kerkeni, A.; and Beguir, K. 2018. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization. *arXiv preprint arXiv:1807.01672*.

Longstaff, F. A., and Schwartz, E. S. 2001. Valuing american options by simulation: a simple least-squares approach. *The review of financial studies* 14(1):113–147.

Malevergn, Y., and Sornette, D. 2006. *Extreme financial risks: From dependence to risk management*. Springer Berlin Heidelberg.

Markowitz, H. 1952. PORTFOLIO SELECTION. *The Journal of Finance* 7(1):77–91.

Michaud, R. O. 2008. Efficient asset management: A practical guide to stock portfolio optimization and asset. *Oxford University Press.*

Perrin, S., and Roncalli, T. 2019. Machine learning optimization algorithms , portfolio allocation. *ssrn preprint ssrn:3425827*.

Powell, W. B. 2011. Approximate dynamic programming: Solving the curses of dimensionality. *John Wiley & Sons*.

Richard, J.-C., and Roncalli, T. 2019. Constrained risk budgeting portfolios: Theory, algorithms, applications & puzzles. *arXiv preprint arXiv:1902.05710*.

Salas, D. F., and B, W. 2013. Powell. benchmarking a scalable approximation dynamic programming algorithm for stochastic control of multidimensional energy storage problems. *Technical report, Princeton University*.

Shi, G.; Shi, X.; O'Connell, M.; Yu, R.; Azizzadenesheli, K.; Anandkumar, A.; Yue, Y.; and Chung, S.-J. 2018. Neural lander: Stable drone landing control using learned dynamics. *arXiv preprint arXiv:1811.08027*.

Wang, Q. Z. . J. W. . R. 2013. Mean-variance asset-liability management with state-dependent risk aversion. Papers 1304.7882, arXiv.org.

Yan Duan, Xi Chen, R. H. J. S., and Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous control. *In Proceedings of The 32nd International Conference on Machine Learning (ICML)*.