Time-Constrained Predictive Modeling on Large and Continuously Updating Financial Datasets

Bernard Lee[1], *Nicos Christofides*[2]

Abstract

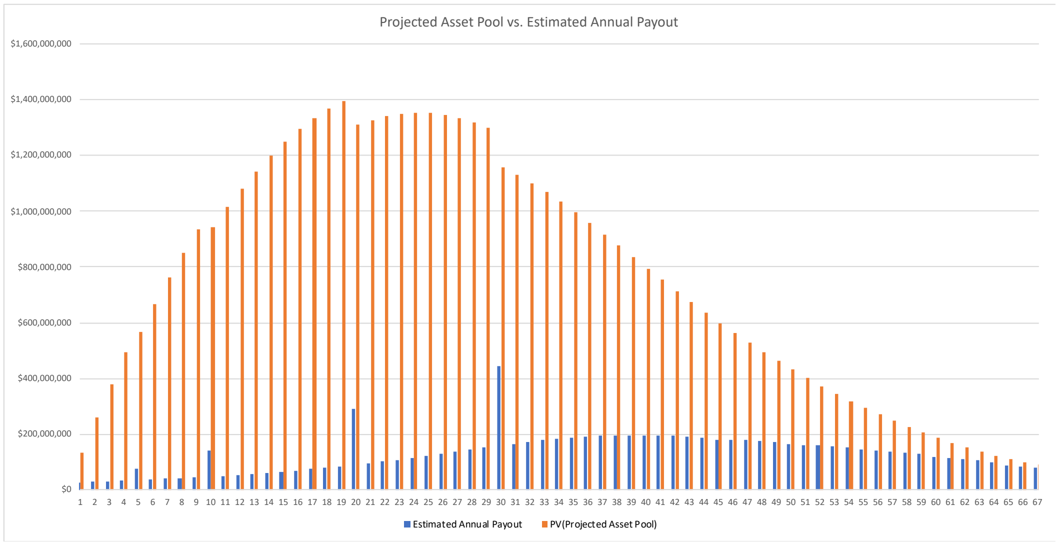
Using artificial neural networks to perform predictive modeling in finance has attracted significant media attention, and pundits are quick to speculate on their potential impacts on jobs and society. With the recent advancement in algorithms and readily accessible, highly specialized hardware, do these algorithms actually live up to such promises? Some unique features of applying artificial neural networks to finance include: 1) Financial datasets change almost constantly; 2) Calculation universe is usually quite large (e.g. a typical sub-universe of over a hundred companies, each of them having several hundred fundamental factors, with many different combinations of time lags); 3) Non-linear statistics that require multiple neurons, layers, and/or calibration with even more specialized techniques; 4) Risk of overfitting models that yield low predictive power; 5) If the computation cannot finish by a specific time constraint (say within 1~2 minutes, but the threshold will be dependent on the specific markets traded) the market may have moved by so much that any modeling results obtained can no longer be helpful to decision making in real-life markets.

The goal of this paper is to provide a formal study on the feasible ranges where predictive power and computational speed-up for different techniques in predictive modeling in finance can be helpful to solving time-constrained algorithms in finance, based on the most up-to-date technology available. The first author presented a similar formal study at the Russian Academy of Sciences in Moscow in 2011, while similar published studies were performed almost a decade ago.[3] We achieve this objective by benchmarking computational performance on a state-of-the-art platform with real-life financial datasets using a 30,000-core supercomputer hosted by the National Supercomputing Centre (NSCC) Singapore.

[1] Founder and Chief Executive, HedgeSPA Limited, 77 Ayer Rajah Crescent #03-36, Singapore 139954. The authors acknowledge generous research support by the National Supercomputing Centre (NSCC) Singapore.
[2] Professor Emeritus, Imperial College London, Exhibition Road, London, UK SW7 2AZ.
[3] See, for instance, A. Bahrammirzaee, "A Comparative Survey of Artificial Intelligence Applications in Finance: Artificial Neural Networks, Expert System and Hybrid Intelligent Systems", *Neural Computing and Applications*, Springer-Verlag, November 2010, Volume 19, Issue 8, pp 1165-1195.

Brief Overview of HedgeSPA Core Investment Platform

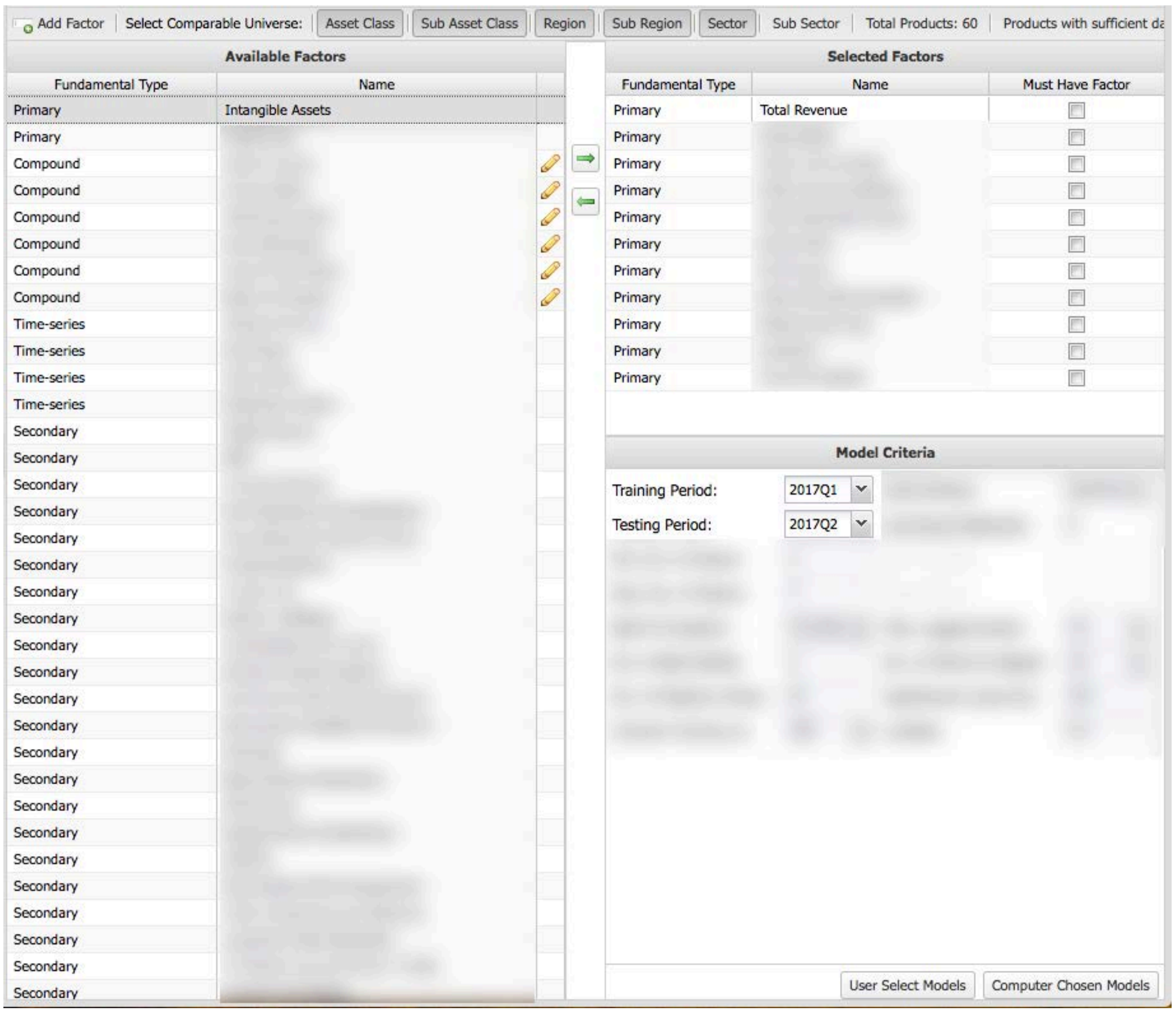
The HedgeSPA platform is a core investment platform that allows institutional investors to perform automated tasks such as asset selection, portfolio rebalancing, decision/execution and reporting. It was cited by Diligence Vault[4] as a leading player in the global InvestTech ecosystem in 2018. The platform serves four industry verticals: Asset Management, Insurance, Wealth Management and Energy. For instance, the platform can mathematically perform a vigorous calculation for insurance clients by estimating insurance pool surplus from liability profiles and actuarial tables, such as the following:



Problem Definition

One key step in client calculations is asset selection driven by machine learning. The HedgeSPA Platform crawls the internet and any other available data sources to find all relevant factors related to an asset universe (e.g. stocks in the US healthcare sector). Typical factors may include company financials, regulatory filings, macroeconomic data, sentiment scores from news and social media, and more specialized information such as production and shipment data related to specific sectors, as shown in the following:

[4] See https://www.linkedin.com/pulse/rise-integrated-investtech-monel-amin/.



Add Factor | Select Comparable Universe: | Asset Class | Sub Asset Class | Region | Sub Region | Sector | Sub Sector | Total Products: 60 | Products with sufficient da

Available Factors

Fundamental Type	Name
Primary	Intangible Assets
Primary	[redacted]
Compound	[redacted]
Compound	[redacted]
Compound	[redacted]
Compound	[redacted]
Compound	[redacted]
Compound	[redacted]
Time-series	[redacted]
Time-series	[redacted]
Time-series	[redacted]
Time-series	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]
Secondary	[redacted]

Selected Factors

Fundamental Type	Name	Must Have Factor
Primary	Total Revenue	☐
Primary	[redacted]	☐
Primary	[redacted]	☐
Primary	[redacted]	☐
Primary	[redacted]	☐
Primary	[redacted]	☐
Primary	[redacted]	☐
Primary	[redacted]	☐
Primary	[redacted]	☐
Primary	[redacted]	☐
Primary	[redacted]	☐

Model Criteria

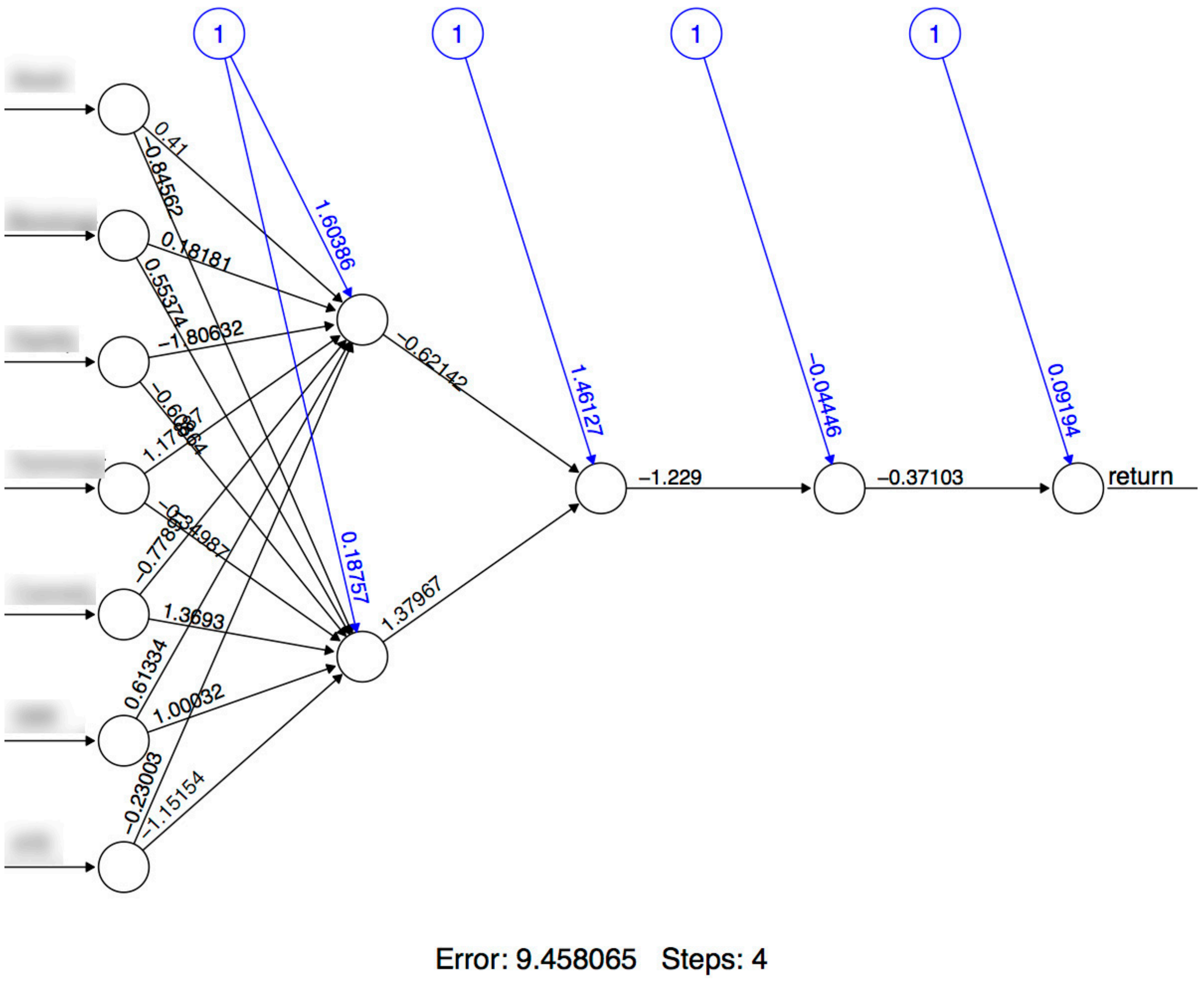
Training Period: 2017Q1

Testing Period: 2017Q2

[redacted]

User Select Models | Computer Chosen Models

Users are given the additional option to filter out irrelevant factors as they see fit. One classic example is how inventory data is not relevant to the internet software sector. Once the choice of factors is confirmed, the computer will try all possible combinations of factors, all possible combination of time frames, as well as all available methodologies in order to find the most predictive model based on the amount of historical data specified for the training. This is analogous to how the `leaps()` function in R performs an exhaustive search for the best subsets of variables in `x` for predicting `y` in linear regression, except for the modification that instead of linear regression, the neural network is used as a form of higher order regression, as shown in the following:



In the case of `leaps()`, an exhaustive search can evolve into step-wise regression. Training and testing neural networks is a time-consuming operation as compared to linear regression, so it is natural to find smart ways to cut down computational time. We will discuss such time saving methods in the final section of this paper.

Proposed Solution

The neural network training and testing algorithm on the HedgeSPA platform has been isolated and parallelized at the National Supercomputing Center in Singapore with the following parameters:

```
Start of Training Period = 2
Testing Period = 4
Lag Period = 1
Number of Factors = 9
Number of Stocks = 60
Min Number of Factors = 2
Max Number of Factors = 5
```

The `Start of Training Period` refers to the quarter when the neural network starts training and the `Testing Period` refers to the quarter for testing. The `Lag Period` represents the number of quarter(s) looking forward in order to predict the future returns of the assets. In this test case, quarters `2` and `3` are used for training, quarter `4` for testing, in order to predict asset returns in quarter `5`. Most other input parameters are self-explanatory.

The total number of subproblems to be solved will be:

```
Num_Combo =
C(Num_Factors, Min_Factors) + … + C(Num_Factors, Max_Factors)
```

The parallelization is done by OpenMP.[5] Since it will be too difficult to migrate the HedgeSPA Platform to any specialized architecture such as a supercomputer, a separate I/O Manager is written in C++ to drive the computational code. The heavy computation is written in C and linked to the Fast Artificial Neural Network Library (FANN) `libfann`.[6] We did not link it to the vectorized `libcudann` library[7] because `libcudann` requires CUDA[8] 3.2, which is much older than the current version 9.x supported by the supercomputer. Rebuilding `libudann` with a more modern version of CUDA will be one of our future research goals.

The specific loop around the parallelized call is implemented as a `for` loop for each training/testing subproblem:

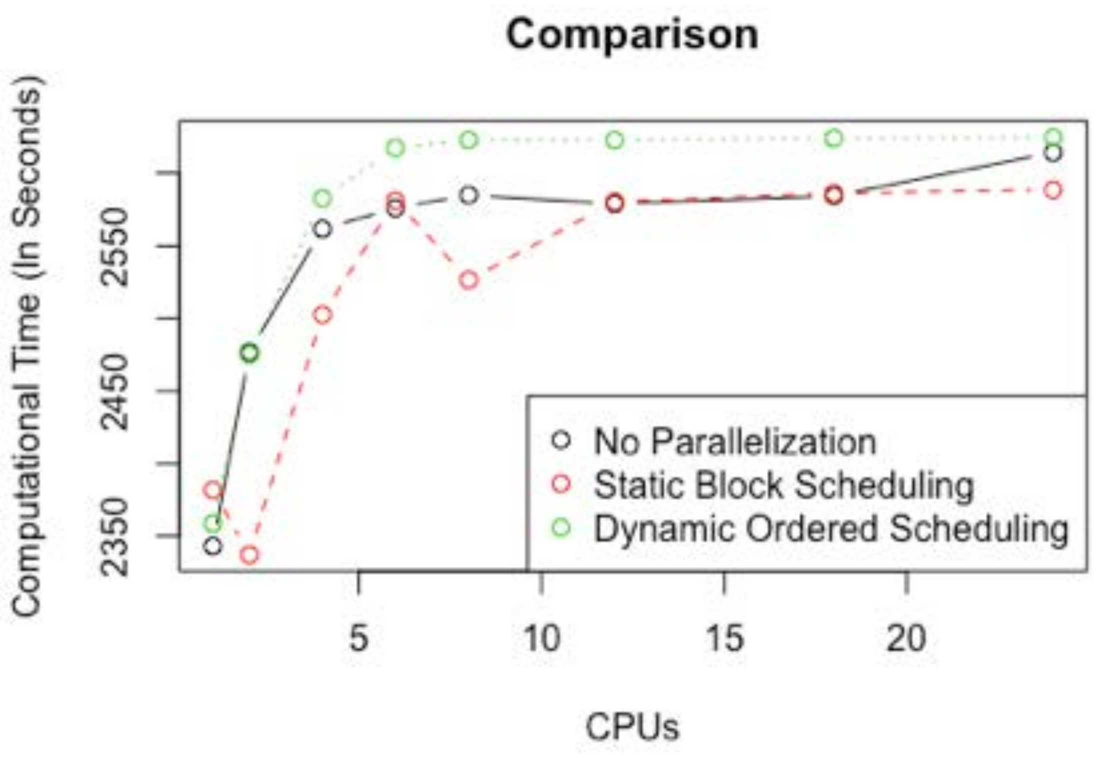
```
#pragma omp parallel private(i_num_model)
{
#pragma omp for
        for(i_num_model = num_model_start; i_num_model <
num_model_end; i_num_model++)
        {
           nn_factor_model_model_selection(i_num_model, r);
//fann computation
       }
}
```

Computational Results

A total of three series of runs were performed based on 1, 2, 4, 5, 8, 12, 18 and 24 CPUs. We cut off the computation at 24 CPUs after no significant speed-up is observed. *No parallelization* was done for the first set of runs. The curly brackets (in

[5] See `https://www.openmp.org/`.
[6] See `https://github.com/libfann`.
[7] See `https://sourceforge.net/projects/libcudann/`.
[8] See `https://developer.nvidia.com/cuda-toolkit`.

RED) in the code above are accidentally omitted, which means that supercomputer can still parallelizes the code. However, each processor is running the same set of code as if each of them is running the full computation on a single CPU. This turns out to be a good control set in that the I/O time for consolidating the results with no computation speed-up will be correctly captured. The second set of runs is based on *Static Block Scheduling* in which the tasks are handed out based on a static allocation algorithm. The final set of runs is based on dynamic ordered scheduling in which the tasks are handed out on a round-robin basis, but the start and completion must be ordered. Sometimes doing so may reduce I/O contention among processors. The completion times of these computational runs are shown in the following:



Static Block Scheduling is the best performing algorithm, but it still shows a rather limited speedup beyond the first few processors. *Dynamic Ordered Scheduling* is worse than *No Parallelization* given the amount of I/O overhead involved.

Lessons Learned

This problem appears to be a classic embarrassingly parallel problem since we are feeding relatively small (by industry standards) sub-problems in a universe of US healthcare stocks to the neural network. However, the nature of financial data on an industrial scale is proven to create too much I/O contention and overhead to achieve reasonable speed up under a multi-CPU architecture. While it may be possible to find a smart workaround with classic, low-level, master-to-slave message passing set up, such a technique is usually done on a one-off basis for scientific applications. Such an architecture is unlikely to be fail-safe if it is called a hundred times just to compare one hundred different asset universes for a typical single-iteration institutional run. In addition, our parallel code is written by parallel programmers with decades of related research experience. Without using a common parallel protocol such as OpenMP, it will be nearly impossible to keep up with the day-to-day maintenance of such parallel code for any practical industry application.

CUDA/GPU and/or other forms of hardware acceleration reportedly give speed-ups of 50 to 100 times to train and test neural networks, without incurring the cost of very large CPU-based clusters. Clearly, these results show that the best value added per research dollar spent is to focus on using a few powerful GPUs on a grid architecture, and which is in fact sold today by a number of vendors.

Future Research

This neural network training/testing problem is essentially a form of combinatoric optimization. The goal is to either minimize time to solution or maximize the probability of finding the best available approximation within a limited timeframe.

Due to hardware technology limitation, our future research goal is to minimize the number of subproblems used and therefore I/O contention. This is algorithm design driven by the need to optimize available hardware technology.

Traditional conjugate gradient style methodologies are unlikely to be effective since there is no reasonable basis to think that any gain function from one neural network to the next (despite a small change in factor configuration) will be continuous. Instead, our goal is to apply graph-theoretic techniques since they are proven in successfully solving practical combinatoric optimization problems.[9]

The authors do wish to caution that this line of research does require significant research funding to pay for access to financial data, supercomputer time, and technologists with financial industry experience. The mathematics and detailed empirical results from the complex financial analytics problems discussed by this paper can be found in a forthcoming title.[10]

[9] N. Christofides, *Graph Theory: An Algorithm Approach*, Academic Press, 1975.
[10] B. Lee, *Investment Analytics: Theory and Practice*, World Scientific, 2018.