GENETIC NEURAL NETWORKS

AUTHOR: THEO DIMITRASOPOULOS* | ADVISOR: ZACHARY FEINSTEIN*

* Department of Financial Engineering; Stevens Institute of Technology Babbio School of Business

ABSTRACT — This paper focuses on the training procedure of artificial neural networks for portfolio management. More specifically, it investigates a neural network -with weights derived from a genetic algorithm embedded in the training process, that automates a buy/sell/hold procedure by learning from the direction of a stock within a "diversified portfolio" (in this case a portfolio of the Dow Jones Industrial Index; hence the loose usage of the term). The binary indicators are generated using a function that observes the daily percentage changes of the index and assigns the corresponding operator if the change is a more than 5% gain, more than a 5% loss, or anything in between. Further development of the project can incorporate a diversified portfolio of securities and reallocate the portfolio weights and/or target beta and then execute the appropriate transactions or complete liquidation of the portfolio depending on its performance. While genetic algorithms are not the end-all of metaheuristic methods, the healthy skepticism that machine learning in a panacea for quantitative finance warrants a rethinking of neural networks. Embedding more traditional techniques could provide better alternatives to the sometimes unnecessarily -and even fatally, complex methods developed in the field.

KEYWORDS — *Algorithms, Artificial, Evolutionary, Genetic, Liquidation, Networks, Neural, Portfolio Management, Training.*

I. FEEDFORWARD NEURAL NETWORKS

A deep feedforward network is a function of the form,

$$\boxed{\hat{\boldsymbol{Y}}(\boldsymbol{X}) := \boldsymbol{F}_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{X}) = (\boldsymbol{f}^{(\boldsymbol{L})}_{\boldsymbol{W}^{(\boldsymbol{L})},\boldsymbol{b}^{(\boldsymbol{L})}} \dots \circ \boldsymbol{f}^{(\boldsymbol{1})}_{\boldsymbol{W}^{(\boldsymbol{1})},\boldsymbol{b}^{(\boldsymbol{1})}})(\boldsymbol{X})}$$

where,

- $f^{(l)}_{W^{(l)},b^{(l)}}(X) := \sigma^{(l)}\left(W^{(l)}X + b^{(l)}\right)$ is a semi-affine function with $\sigma^{(l)}$ as a univariate, continuous non-linear activation function such as $tanh(\bullet)$, $sigmoid$, or $max(\bullet, 0)$,
- $W = \left(W^{(1)}, \dots, W^{(L)}\right)$ are the weight matrices (which will be substituted by the optimal weights generated by the genetic algorithm described below),
- $b = \left(b^{(1)}, \dots, b^{(L)}\right)$ are the offsets.

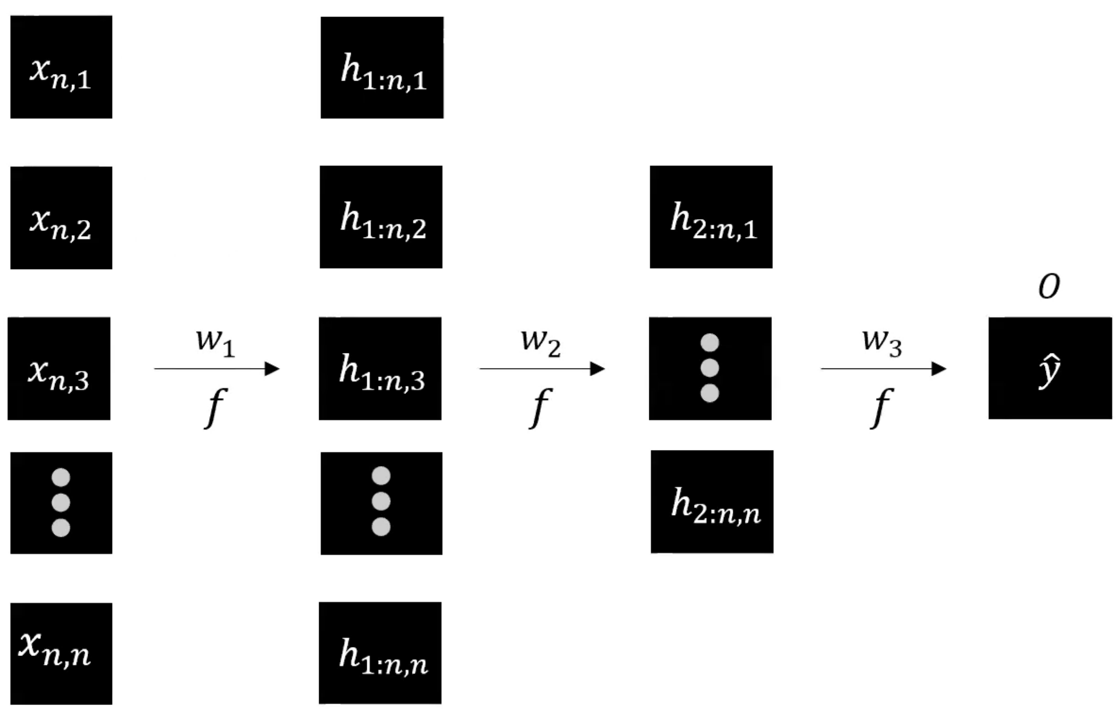

Figure 1: Structure of a feedforward network.

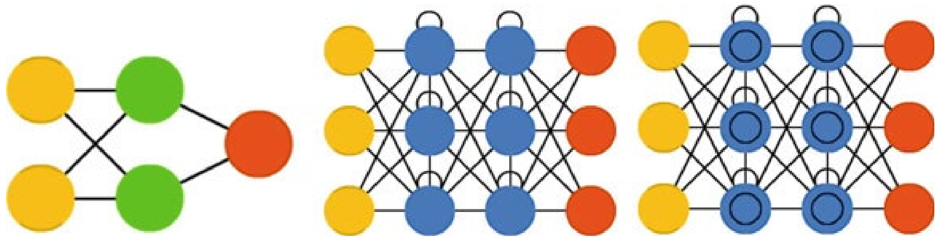

Figure 2: A feedforward, a recurrent and a Long-Short Term Memory (LSTM) network. Yellow indicates the input nodes, blue the hidden nodes with recurrence/memory, and red the output nodes. Source: Van Veen, F. & Leijnen, S. (2019)

II. GENETIC ALGORITHMS

A. Survival of the fittest

Genetic algorithms search for the optimal solution in a solution space. While Darwinian evolution maintains a population of specimens, genetic algorithms work through a population of candidate solutions, called individuals. The candidate solutions are evaluated and transformed into a novel generation of solutions. In a "survival of the fittest" contest, the more suitable solutions have an increased probability of being selected and passing their qualities to the next generation of candidate solutions.

B. Population

Since each individual solution is usually represented by a series of binary strings, the population of individuals can be viewed as a collection of solutions. The population represents the current population and evolves with each iteration

C. Fitness function

At each iteration of the algorithm, the individuals are evaluated using a fitness or target function, which is the function under optimization. Individuals with a higher fitness score are more likely candidates to create "offspring" in the next generation of solutions. Over time, the quality of the solutions improves, the fitness values increase, and the process can stop once a solution is found with a satisfactory fitness value.

D. Crossover

For the next generation, two parents are chosen from the current generation and are recombined to create children weights in a specific fashion. In this project, a quasirandom matrix multiplication between the different weights is implemented.

E. Mutation

A mutation works as a seed to refresh the population and expand the areas under search in the solution space. Mutations in this project were implemented as a multiplication with an integer drawn from a random uniform distribution.

The main advantages of genetic algorithms that make them suitable for a neural network hybrid are the following:

- Potential to find a global optimum
- Ability to handle problems with a complex mathematical definition
- Adversity to noise

III. DATA

Date	^DJI	Percent Change	Liquidate
2007-01-03	12474.519531	0.000000	0
2007-01-04	12480.690430	0.000495	0
2007-01-05	12398.009766	-0.006625	-1
2007-01-08	12423.490234	0.002055	0
2007-01-09	12416.599609	-0.000555	0
...	...	...	...
2020-01-06	28703.380859	0.002392	0
2020-01-07	28583.679688	-0.004170	0
2020-01-08	28745.089844	0.005647	1
2020-01-09	28956.900391	0.007369	1
2020-01-10	28823.769531	-0.004598	0

Figure 3: Sample data used in the training/testing process. It is worth noting that the "Liquidate" column refers to the binary operators assigned to each datapoint depending on whether the movement is favorable for the "diversified portfolio".

The data was generated using a built-in percent change function and a custom if/elif/else function that assigns the binary operators to each datapoint. The daily adjusted close price across 7 years for the Dow Jones Industrial index is also suitable for the discovery of a fundamental beta upon the expansion of the engine source code to include a real diversified portfolio of securities. However, the user is in the position to adjust the duration of the time span of their data to meet their needs.

IV. GENETIC NEURAL NETWORKS

A genetic neural network can be defined as the same function for a neural network,

$$\boxed{\widehat{\boldsymbol{Y}}(\boldsymbol{X}) := \boldsymbol{F}_{\text{🧬},\boldsymbol{b}}(\boldsymbol{X}) = (\boldsymbol{f}^{(\boldsymbol{L})}_{\text{🧬}^{(\boldsymbol{L})},\boldsymbol{b}^{(\boldsymbol{L})}} \cdots \circ \boldsymbol{f}^{(\boldsymbol{1})}_{\text{🧬}^{(\boldsymbol{1})},\boldsymbol{b}^{(\boldsymbol{1})}})(\boldsymbol{X})}$$

where,

- $f^{(l)}_{\text{🧬}^{(l)},b^{(l)}}(X) := \sigma^{(l)}\left(\text{🧬}^{(l)}X + b^{(l)}\right)$ is a semi-affine function with $\sigma^{(l)}$ as a univariate, continuous non-linear activation function such as $tanh(\bullet)$, $sigmoid$, or $max(\bullet, 0)$,
- $\text{🧬} = \left(\text{🧬}^{(1)}, \dots, \text{🧬}^{(L)}\right)$ are the weight matrices (which will be substituted by the optimal weights generated by the genetic algorithm described below,
- $b = \left(b^{(1)}, \dots, b^{(L)}\right)$ are the offsets.

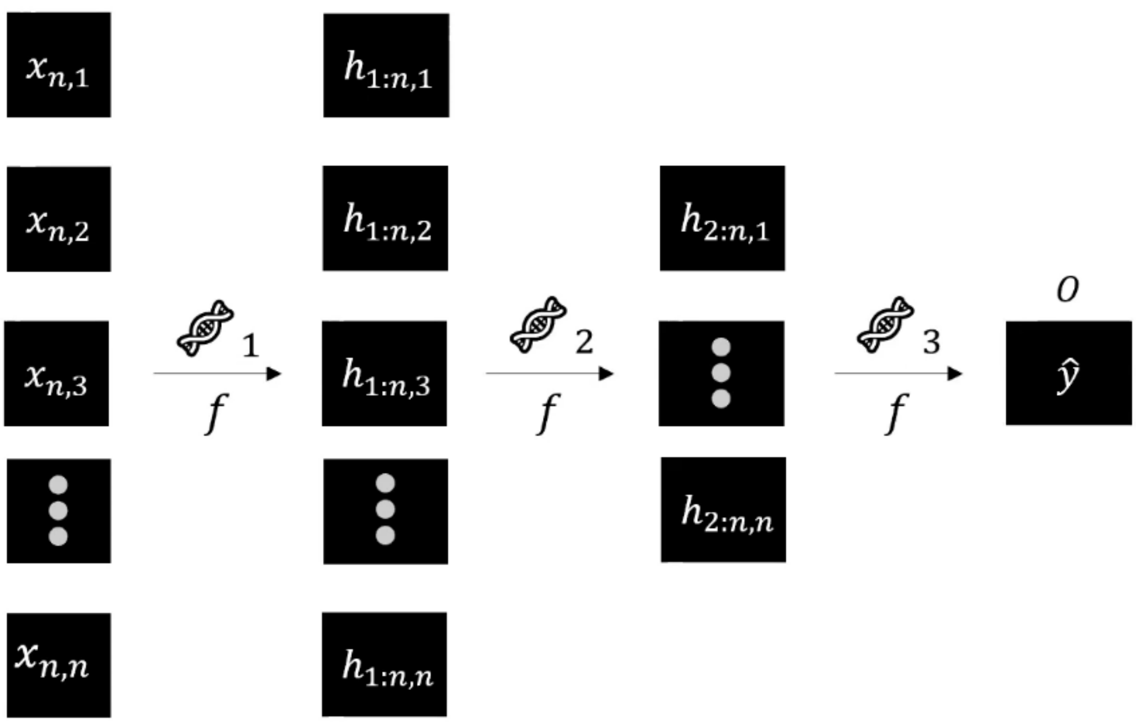

Figure 4: Structure of a genetic neural network.

V. RESULTS AND DISCUSSION

The parameters and essential metrics of the Feedforward Neural Network construct are the following:

- 500 Epochs
- Initial Fitness Score: 0.26
- Fitness Score at the end of Epoch 500: 0.53

```
Epoch 1/500 77/77 [==============================]   0s 3ms/step
- loss: 0.9712 - accuracy: 0.2590
Epoch 2/500 77/77 [==============================]   0s 3ms/step
- loss: 0.9698 - accuracy: 0.2590
Epoch 3/500 77/77 [==============================]   0s 3ms/step
- loss: 0.9685 - accuracy: 0.2590
Epoch 4/500 77/77 [==============================]   0s 3ms/step
- loss: 0.9672 - accuracy: 0.2590
...
Epoch 497/500 77/77 [==============================] 0s 3ms/step
- loss: 0.7410 - accuracy: 0.5474
Epoch 498/500 77/77 [==============================] 0s 3ms/step
- loss: 0.7410 - accuracy: 0.5474
Epoch 499/500 77/77 [==============================] 0s 3ms/step
- loss: 0.7410 - accuracy: 0.5474
Epoch 500/500 77/77 [==============================] 0s 3ms/step
- loss: 0.7410 - accuracy: 0.5474
Test Accuracy: 0.53
```

Figure 5 Sample output of the Feedforward Neural Network.

In contrast, the following are the parameters and essential metrics of the Genetic Neural Network:

- 10 Epochs
- Initial Fitness Score: 0.52
- Fitness Score at Generation 160: 0.81

We see that the test accuracy increased by 28% within a shorter relative timespan. However, the absolute running time of the genetic algorithm was 12% higher, owing to the increased computational complexity of the genetic algorithm. The metaheuristic could be further improved with a more elaborate penalization mechanism in the mutation step (than the one currently implemented) to test for separate cases and exclude more scenarios where the mutations themselves are deemed unsuitable by the user.

```
[array([[1.4918541]], dtype=float32), array([[-
0.9948138 , 0.36493468, -0.271497 ]], dtype=float32),
array([[-0.44647908, 0.42858624, -0.00314045], [-
0.14004254, 0.7530701 , 0.5343478 ], [ 0.28062987, -
0.89855146, -0.8241141 ]], dtype=float32),
array([[-0.27276278, 0.20729327, -0.39682007], [
0.5404234 , -0.19788313, 0.4285028 ], [ 0.10512328, -
0.47381616, -0.66573787]], dtype=float32), array([[-
0.32383364], [-1.1936888 ], [-1.0405163 ]],
dtype=float32)]
Gen 1 Test Accuracy: 0.52
```

Figure 6 Sample output of the Genetic Neural Network.

VI. NEXT STEPS

In the next steps of this project, one can create pipeline for portfolios of multiple securities to fully implement the concept of the genetic neural network. The signals could then be based on realized portfolio return and reallocate portfolio weights and/or target beta at specified time intervals (daily, weekly etc.). Finally, the algorithm can then execute the buy/hold/sell of individual securities or liquidation of the entire portfolio depending on the performance of its constituents, as the current design stands.

VII. ACKNOWLEDGMENTS

I would like to thank Professor Feinstein (or Zach, given the comfort and safety he imbued in the environment he facilitated for us students) for his continued support in identifying the pain points and potential developments in the project proposal and throughout the semester overall. His lessons will be a valuable component of my skill set in the future and I will always refer to them during my professional career; The Stevens Institute of Technology Department of Financial Engineering for its provision of laptops and technological equipment to undertake this project. Finally, I would like to thank the developers of the Google Collaboratory project for providing a streamlined platform for the testing and execution of the models.

VIII. REFERENCES

[1] Eyal Wirsansky. Hands-On Genetic Algorithms with Python: Applying genetic algorithms to solve real-world deep learning and artificial intelligence problems. Packt Publishing. ISBN: 1838557741, 978-1838557744. 1st Edition, 2020.

[2] Matthew F. Dixon, Igor Halperin, Paul Bilokon. Machine Learning in Finance: From Theory to Practice. Springer. ISBN: 3030410676,9783030410674. 1st Edition, 2020.

[3] Zachary Feinstein. Notes on Genetic Algorithms. FE 690: Machine Learning in Finance. Stevens Institute of Technology. 2020.

[4] Samer Obeidat, Daniel Shapiro, Mathieu Lemay, Mary Kate MacPherson, Miodrag Bolic. Adaptive Portfolio Asset Allocation with Deep Learning. International Journal on Advances in Intelligent Systems, Vol 11, No 1, 2. 2018 http://www.iariajournals.org/intelligentsystems/. Accessed: November 30th, 2020.

IX. APPENDIX

```
# -*- coding: utf-8 -*-

"""
Automatically generated by Colaboratory.
Original file is located at:
https://colab.research.google.com/drive/13By3WikWhCD1BNPSGeuOV_9OUa3Ww5u3
"""

# Import Packages
import random
import numpy as np
import pandas as pd
from keras.layers import Dense
from keras.models import Sequential
from sklearn.metrics import accuracy_score
from sklearn.model_selection import
train_test_split
import pandas_datareader.data as web
import matplotlib.pyplot as plt

# Genetic Neural Network class
class ENN(Sequential):
    def __init__(self, wts_sub=None):
        super().__init__()
        if wts_sub is None:
            layer1 = Dense(1,
input_shape=(1,), activation='sigmoid')
            layer2 = Dense(3,
activation='sigmoid')
            layer3 = Dense(3,
activation='sigmoid')
            layer4 = Dense(3,
activation='sigmoid')
            layer5 = Dense(1,
activation='sigmoid')

            self.add(layer1)
            self.add(layer2)
            self.add(layer3)
            self.add(layer4)
            self.add(layer5)

        else:

self.add(Dense(1,input_shape=(1,),activation='sigmoid',weights=[wts_sub[0],
np.zeros(1)]))
self.add(Dense(3,activation='sigmoid',weights=[wts_sub[1], np.zeros(3)]))
self.add(Dense(3,activation='sigmoid',weights=[wts_sub[2], np.zeros(3)]))
self.add(Dense(3,activation='sigmoid',weights=[wts_sub[3], np.zeros(3)]))
self.add(Dense(1,activation='sigmoid',weights=[wts_sub[4], np.zeros(1)]))

def f_propagation(self, X_train, y_train):
        y_predicted =
self.predict(X_train.values)
        self.fitness =
accuracy_score(y_train,
y_predicted.round())

def train(self, epochs):
self.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])
        self.fit(X_train.values,
y_train.values, epochs=epochs)

def mutation(wts_sub):
    choose = random.randint(0,
len(wts_sub)-1)
    mut = random.uniform(0, 1)
    if mut >= .5:
        wts_sub[choose] *=
random.randint(2, 5)
    else:
        pass

def cross(nn1, nn2):
    nn1_weights = []
    nn2_weights = []
    wts_sub = []

    for layer in nn1.layers:
nn1_weights.append(layer.get_weights()[0])

    for layer in nn2.layers:
nn2_weights.append(layer.get_weights()[0])

    for i in range(0, len(nn1_weights)):
        split = random.randint(0,
np.shape(nn1_weights[i])[1]-1)

        for j in range(split,
np.shape(nn1_weights[i])[1]-1):
            nn1_weights[i][:, j] =
nn2_weights[i][:, j]
        wts_sub.append(nn1_weights[i])
    mutation(wts_sub)
    child = ENN(wts_sub)
    return child
```

```
# Data processing
prices = pd.DataFrame()
tickers = ['^DJI']

for i in tickers:
    tmp = web.DataReader(i, 'yahoo',
'1/1/2007', '01/12/2020')
    prices[i] = tmp['Adj Close']

prices['Percent Change'] =
prices.pct_change()

def set_signal(column):
    if column['Percent Change'] < -0.0050:
        signal = -1
    elif column['Percent Change'] > 0.0050:
        signal = 1
    else:
        signal = 0
    return signal

prices['Liquidate'] =
prices.apply(set_signal, axis=1)
prices = prices.replace(np.nan,0)
prices = prices.drop(['^DJI'], axis=1)
prices.reset_index(inplace=True,drop=True)

# Split into independent/dependent
variables (training/testing data)
X = prices['Percent Change']
X = 100*X.round(8)
X = X.astype(np.float32)
print(X.astype(np.float32))
y = prices.drop(['Percent Change'], axis=1)
#y = y.astype(np.float32)
print(y)
X_train, X_test, y_train, y_test =
train_test_split(X, y)

# Initiate Genetic Neural Network instance
networks = []
pool = []
gen = 0
n = 20

for i in range(0, n):
    networks.append(ENN())
fit_max = 0
wts_opt = []

while fit_max < .9:
    gen += 1
    print('Generation', gen)

    for nn in networks:
        nn.f_propagation(X_train, y_train)
        pool.append(nn)
    networks.clear()
    pool = sorted(pool, key=lambda x:
x.fitness)
    pool.reverse()

    for i in range(0, len(pool)):

        if pool[i].fitness > fit_max:
            fit_max = pool[i].fitness
            print('Fitness Score: ',
fit_max)
            wts_opt = []

            for layer in pool[i].layers:
          wts_opt.append(layer.get_weights()[0
])
            print(wts_opt)

    for i in range(0, 5):

        for j in range(0, 2):
            temp = cross(pool[i],
random.choice(pool))
            networks.append(temp)

portfolio_enn = ENN(wts_opt)
portfolio_enn.train(10)
y_predicted =
portfolio_enn.predict(X_test.values)

print('Accuracy: %.2f' %
accuracy_score(y_test,
y_predicted.round()))

# Feedforward neural network as a benchmark
portfolio_nn = Sequential()
portfolio_nn.add(Dense(1, input_shape=(1,),
activation='sigmoid'))
portfolio_nn.add(Dense(3,
activation='sigmoid'))
portfolio_nn.add(Dense(3,
activation='sigmoid'))
portfolio_nn.add(Dense(3,
activation='sigmoid'))
portfolio_nn.add(Dense(1,
activation='sigmoid'))

portfolio_nn.compile(optimizer='rmsprop',lo
ss='hinge',metrics=['accuracy'])

portfolio_nn.fit(X_train.values,
y_train.values, epochs=500)
```

```
y_predicted =
portfolio_nn.predict(X_test.values)
y_predicted = np.around(y_predicted, 0)

print('Test Accuracy: %.2f' %
accuracy_score(y_test,
y_predicted.round()))
```