
Graduation Thesis

# A unified Machine Learning / Deep Learning framework for security trading and predicting traders' strategies

Nguyen Duc Linh*

Supervisor: Dr.Nguyen Chi Kien

**John von Neumann Institute, Vietnam**

June 10, 2020

**Abstract**

Traditionally, individual and institutional traders have been employing fundamental analysis, technical analysis, and sentiment analysis to make portfolio decisions (e.g., to buy, hold, or sell a security). It has been shown that in some situations, Machine Learning (ML) models provide more flexible and accurate predictions than traditional methods. Previous work in this area leverages various models in ML such as Support Vector Machine (SVM), Random Forests, Gradient Boosting, and Reinforcement Learning (RL), and in Deep Learning (DL) such as Recurrent Neural Network (RNN), and Long Short-Term Memory (LSTM) to predict the price of a security or a group of related securities. The features used for these models can be derived from fundamental factors, technical factors, and sentimental factors. Natural Language Processing (NLP) has also been used to extract relevant information and features from social media and news. DL has been used to compute relevant features from raw features. Many researchers have also employed RL to compute the optimal strategies at each state, given market data and previous returns. Also, from the optimal strategies, we can predict the strategies of rational traders in the market. In this research, we will design a unified framework using DL and RL to extract features from fundamental, technical, and sentimental factors and compute the optimal strategies in security trading to predict the strategies and behaviors of rational traders in the market. The framework will be tested by applying to Vietnam's stock market and predict the strategies and behaviors of traders.

***Keywords:*** DL, financial signal processing, Neural Network (NN) for finance, RL

*linh.nguyen2018@qcf.jvn.edu.vn

# Contents

# List of Figures



# List of Tables

# Symbol

$b_j^{(i)[k]}$ value of the bias for $j^{th}$ hidden unit in $i^{th}$ training example in layer $k$

$f_\theta$ decoder function

$g_\phi$ encoder function

$m$ number of training examples

$\sigma$ sigmoid function

$w_j^{(i)[k]}$ value of the weight for $j^{th}$ hidden unit in $i^{th}$ training example in layer $k$

$x_j^{(i)[k]}$ value of the feature $j$ in $i^{th}$ training example in layer $k$

$\hat{x}_j^{(i)[k+5]}$ predicted value $j$ from $i^{th}$ training example in layer $k+5$

$z_l^{(i)[k+1]}$ value of the feature $l$ in $i^{th}$ training example in layer $k+1$

$z_m^{(i)[k+2]}$ value of the feature $m$ in $i^{th}$ training example in layer $k+2$

$z_n^{(i)[k+3]}$ value of the feature $n$ in $i^{th}$ training example in layer $k+3$

$z_o^{(i)[k+4]}$ value of the feature $o$ in $i^{th}$ training example in layer $k+4$

# Nomenclature

**AE** Autoencoder

**BPTT** Back-Propagation Through Time

**DNN** Deep Neural Network

**DR** Direct Reinforcement

**FDRNN** Fuzzy Deep Recurrent Neural Network

**FNN** Feedforward Neural Network

**HTML** Hypertext Markup Language

**LSTM** Long Short-Term Memory

**MDP** Markov Decision Processes

**MSE** Mean Squared Error

**NLP** Natural Language Processing

**PCA** Principal Component Analysis

**RNN** Recurrent Neural Network

**SVM** Support Vector Machine

**t-SNE** t-Distributed Stochastic Neighbor Embedding

# Abbreviation

**DL** Deep Learning

**EPS** Earning per share

**FLS** Fuzzy Logic System

**ML** Machine Learning

**NN** Neural Network

**PDF** Portable Document Format

**QFS** Quarterly Financial Statement

**RL** Reinforcement Learning

**UL** Unsupervised Learning

**VNM** Vinamilk

# 1 Fuzzy Neural Network

Chin-Teng Lin and Geogre Lee proposed a general neural network (connectionist) model for fuzzy logic control and decision system [LL91] in the form of feedforward multiplayer net, combining the idea of fuzzy logic controller and neural network. A fuzzy logic control/decision network is constructed automatically by learning the examples itself. In this connectionist structure, the input and outputs nodes represent the input state and output/control signal, respectively, and the hidden layers function as membership function and rules.

## 1.1 Fuzzy Logic System

Fig. 1 shows the basic architecture of a traditional Fuzzy Logic System (FLS) with a learning/adapting component. If the output of the defuzzier feeds into the plant and act as a command, it functions as a fuzzy logic controller; otherwise, it functions as a fuzzy logic decision/diagnosis system.

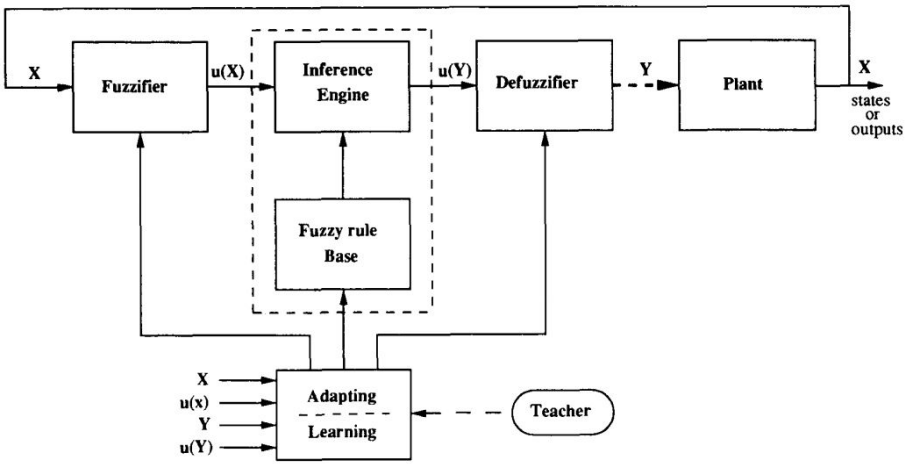


Figure 1: General model of fuzzy logic controller and decision making (diagnosis) system [LL91]

A fuzzy set $F$ in a universe of discourse $U$ is characterized by a membership function $\mu_F : U \mapsto [0,1]$. Thus, a fuzzy set $F$ in $U$ may be represented as a set of ordered pair. Each pair consists of a generic element $u$ and its grade of membership function; that is, $F = \{(u, \mu_F(u))|u \in U\}$. $u$ is called a support value if $\mu_F > 0$. If $U$ is a continuous universe and F is normal and convex (i.e., $max_{u \in U} \mu_F(u) = 1$ and $\mu_F(\lambda u_1 + (1-\lambda)u_2) \geq min(\mu_F(u_1), \mu_F(u_2)), u_1, u_2 \in U, \lambda \in [0,1]$), then $F$ is a *fuzzy number*.

A linguistic variable $x$ in a universe of discourse $U$ is characterized by $T(x) = \{T_x^1, T_x^2, ..., T_x^k\}$ and $M(x) = \{M_x^1, M_x^2, ..., M_x^k\}$, where $T(x)$ is the *term set* of $x$; that is, the set of names of linguistic values of $x$ with each value $T_x^i$ defined on $U$.

The input vector $\mathbf{X}$ which includes the input state linguistic variables $x_i$'s and output state vector $\mathbf{Y}$ which includes the output state linguistic variables $y_i$'s can be defined as

$$\mathbf{X} = \{(x_i, U_i, \{T_{x_i}^1, T_{x_i}^2, ..., T_{x_i}^k\}, \{M_{x_i}^1, M_{x_i}^2, ..., M_{x_i}^k\})|_i = 1...n\} \quad (1)$$

$$\mathbf{Y} = \{(y_i, U_i^{'}, \{T_{y_i}^1, T_{y_i}^2, ..., T_{y_i}^k\}, \{M_{y_i}^1, M_{y_i}^2, ..., M_{y_i}^k\})|_i = 1...m\} \quad (2)$$

The fuzzier in Fig. 1 is mapping from an observed input space to fuzzy sets in a certain input universe of discourse.

The fuzzy rule base in Fig. 1 contains a set of fuzzy logic rules $R$. For a multiinput and multioutput (MIMO) system: $R = \{R_{MIMO}^1, R_{MIMO}^2, ..., R_{MIMO}^n\}$, where the $i^{th}$ fuzzy logic rule is $R_{MIMO}^i$ = IF ($x_1$ is $T_{x_1}$ and ... and $x_p$ is $T_{x_p}$) THEN ($y_1$ is $T_{y_1}$ and ... and $y_q$ is $T_{y_q}$)

The preconditions of $R^i_{MIMO}$ form a fuzzy set $T_{x_1} \times ... \times T_{x_p}$ and the consequence of $R^i_{MIMO}$ is the union of $q$ independent outputs. So the rule can be represented by a fuzzy implication: $R^i_{MIMO} = (T_{x_1} x ... x T_{x_p}) \rightarrow (T_{y_1} + ... + T_{y_q})$, where " + " represents the union of independent variables.

The inference engine in Fig. 1 is to match the preconditions of rules in the fuzzy rule base with the input state linguistic terms and perform implication. For example, if there were two rules:

$$\begin{array}{c} \text{R1: IF } x_1 \text{ is } T^1_{x_1} \text{ and } x_2 \text{ is } T^1_{x_2}, \text{ THEN } y \text{ is } T^1_y \\ \text{R2: IF } x_1 \text{ is } T^2_{x_1} \text{ and } x_2 \text{ is } T^2_{x_2}, \text{ THEN } y \text{ is } T^2_y \end{array}$$

Then the firing strengths of rules $R_1$ and $R_2$ are defined as $\alpha_1$ and $\alpha_2$, respectively. Here $\alpha_1$ is defined as

$$\alpha_1 = M^1_{x_1}(x_1) \wedge M^1_{x_2}(x_2) \tag{3}$$

where $\wedge$ us the *fuzzy AND* operation. Hence, $\alpha_i, i = 1, 2$ becomes

$$\alpha_i = M^i_{x_1}(x_1) \wedge M^i_{x_2}(x_2) = \begin{cases} \min\left(M^i_{x_1}(x_1), M^i_{x_2}(x_2)\right) \\ \text{or} \\ M^i_{x_1}(x_1) M^i_{x_2}(x_2) \end{cases} \tag{4}$$

The above two rules, $R_1$ and $R_2$, lead to the corresponding decision with the membership function, $\hat{M}^i_y(w), i = 1, 2$, which is defined as

$$\hat{M}^i_y(w) = \alpha_i \wedge M^i_y(w) = \begin{cases} \min\left(\alpha_i, M^i_y(w)\right) \\ \text{or} \\ \alpha_i M^i_y(w) \end{cases} \tag{5}$$

where $w$ is the variable that represents the support values of the membership function. Output decision are obtained by combining the above two decisions

$$\hat{M}_y(w) = \hat{M}^1_y(w) \vee \hat{M}^2_y(w) \tag{6}$$

where $\vee$ is the *fuzzy OR* operation). Hence the output decision becomes

$$\begin{aligned} \hat{M}_y(w) &= \hat{M}^1_y(w) \vee \hat{M}^2_y(w) \\ &= \begin{cases} \max\left(\hat{M}^1_y(w), \hat{M}^2_y(w)\right) \\ \text{or} \\ \min\left(1, \left(\hat{M}^1_y(w) + \hat{M}^2_y(w)\right)\right) \end{cases} \end{aligned} \tag{7}$$

Before feeding the signal to the plant, a defuzzification process is needed to get a crisp decision, and the defuzzifier block in Fig. 1 servers this purpose. Among the commonly used defuzzification strategies (i.e center of sums, centroid an mean of maxima method), *center of area* method yields a superior result [BR78]. Let $w_j$ be the support value at which the membership function, $\hat{M}^j_y(w)$, reaches the maximum value $\hat{M}^j_y(w)\Big|_{w=w_j}$, then the fuzzification output is

$$y = \frac{\sum_j \hat{M}^j_y(w_j) w_j}{\sum_j \hat{M}^j_y(w_j)} \tag{8}$$

To provide learning/adapting capability in the traditional fuzzy logic control/decision structure, the connectionist model is proposed

## 1.2 Connectionist Fuzzy Logic Control and Decision System

With the proposed structure Fig. 2, the system are defined with total of five layers. Nodes at layer one are input nodes (linguistic nodes) which represent input linguistic variables. Layer five is the output layer. We have two linguistic nodes for each output variable, one is for training

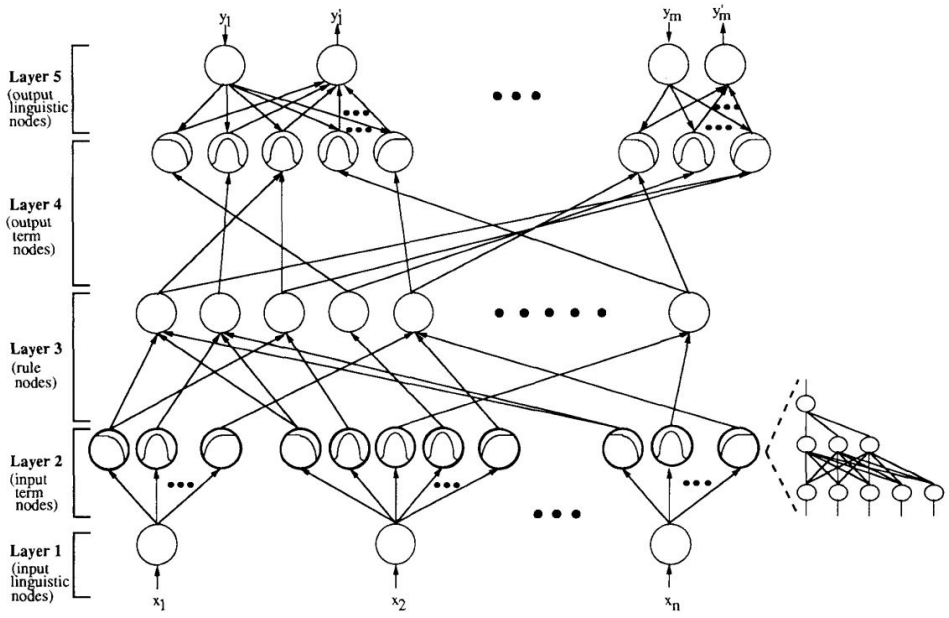


Figure 2: Connectionist fuzzy logic control/decision system [LL91]

data and the other is for actual output. Nodes at layers two and four are *term nodes* of the respective linguistic variable. Each node at layer three is a rule node which represents one fuzzy rule. Net input for each layer are defined as following

$$\text{net-input } = f\left(u_1^k, u_2^k, \cdots, u_p^k; w_1^k, w_2^k, \cdots, w_p^k\right) \tag{9}$$

where the superscript indicates the layer number. A *second action* of each node is to output an activation value as a function its net input

$$\text{output } = o_i^k = a(f) \tag{10}$$

where $a(\cdot)$ denotes the activation function. For example:

$$f = \sum_{i=1}^{p} w_i^k u_i^k \text{ and } a = \frac{1}{1+e^{-f}} \tag{11}$$

Functions of the nodes in each of the five layers are defined as following:

- *Layer 1:* The nodes transmit the input values to the next layer directly with the link weight $w_i^1$ is unity
$$f = u_i^1 \text{ and } a = f \tag{12}$$
- *Layer 2:* Performing a simple membership function, e.g for a bell-shaped function
$$f = M_{x_i}^j\left(m_{ij}, \sigma_{ij}\right) = -\frac{\left(u_i^2 - m_{ij}\right)^2}{\sigma_{ij}^2} \text{ and } a = e^f \tag{13}$$
where $m_{ij}$ and $\sigma_{ij}$ are the center (or mean) and the width (or variance) of the $j^{th}$ term of the $i^{th}$ input linguistic variables $x_i$
- *Layer 3:* Performing precondition matching of fuzzy logic rule *AND* operation with the link weight $w_i^3$ is unity
$$f = \min\left(u_1^3, u_2^3, \cdots, u_p^3\right) \text{ and } a = f \tag{14}$$

- *Layer 4:* Performing two operations: *down-up* transmission and *up-down* transmission. In the *down-up* transmission node, fuzzy *OR* operation is performed to integrate the fired rules with the link weight $w_i^4$ is unity

$$f = \sum_{i=1}^{p} u_i^4 \text{ and } a = \min(1, f) \tag{15}$$

  In the *up-down* transmission, the nodes and links in layer five function exactly the same as those in layer two except that only a single node is used to perform a membership function for output linguistic variable

- *Layer 5:* there are two kinds of nodes in this layer. The first performs the up-down transmission for the training data to feed into the network

$$f = y_i \text{ and } a = f \tag{16}$$

  The second kind of node performs the down-up transmission for the decision signal output. If $m_{ij}^5$ 's and $\sigma_{ij}^5$ 's are the centers and the widths of the membership functions, then following function can be used to simulate the *center of area* defuzzification method

$$f = \sum w_{ij}^5 u_i^5 = \sum \left(m_{ij}\sigma_{ij}\right) u_i^5 \text{ and } a = \frac{f}{\sum \sigma_{ij} u_i^5} \tag{17}$$

  Here the link weight $w_i^5$ is $m_{ij}\sigma_{ij}$

## 1.3 Practical Financial Application of Fuzzy Neural Network

Fuzzy logic has already been applied to a wide range of areas within the field of finance. However, it is far from reaching its full potential compared with its use in other fields, such as control systems, engineering and environmental sciences [SROASP19]. Chiung-Hon Leon Lee, Alan Liu and Wen-Sung Chen proposed a method for candlestick pattern discovery of fuzzy financial time series to predict the stock price or future index [CHLLC06]. Especially, fuzzy learning is an ideal paradigm to reduce the uncertainty in the original data [KF88, PB94], whilst financial signal data has many uncertainties. Fuzzy uses linguistics terms corresponding fuzzy membership degrees to the input data, such change of representation makes the learning system robust with decision control [YDD16]. In detail, the fuzzy rough set can be defined on the increasing, decreasing and no trend group. Mathematically, we define $p_1, p_2, \ldots, p_t, \ldots$ as the price sequences released from the exchange center. Then, the return at time point $t$ is easily determined by $r_t = p_t - p_{t-1}$, the $i^{th}$ fuzzy membership function $v_i(\cdot) : R \to [0, 1]$ maps the $i^{th}$ input $a_i^{(l)}$ of the feature vector $\mathbf{f}_t = [r_{t-m+1}, \ldots, r_t] \in \mathbb{R}^m$, the recent number of training examples $(m)$ return values, as a fuzzy degree

$$o_i^{(l)} = v_i\left(a_i^{(l)}\right) = e^{-\left(a_i^{(l)} - m_i\right)^2 / \sigma_i^2} \quad \forall i \tag{18}$$

where the parameter the fuzzy centers $(m_i)$ and width $\left(\sigma_i^2\right)$ of the fuzzy nodes are initialized using K-Means with fixed k=3, because each input node is connected with three membership functions. Then, in each cluster, the mean and variance of each dimension on the input vector $(\mathbf{f}_t)$ are sequentially calculated to initialize the corresponding $m_i$ and $\sigma_i^2$

# 2 Auto-Encoder

The idea of Autoencoder (AE) was originated in the 1980s, AE was proposed as a method for unsupervised pre-training ad used to learn a mapping in an AE stack pre-trained by this type of Unsupervised Learning (UL) [Bal87, Sch15]

AE is a special type of Feedforward Neural Network (FNN) where the input is the same as the output, it is designed to learn an identity function in an unsupervised way to reconstruct the original input. The network *compresses* the input into a lower-dimensional bottleneck layer and then reconstruct the output from this representation so as to discover a more efficient and compressed representation. The bottleneck layer is a compact *summary* or *compression* of the input, also called the *latent-space representation*

AE has following important properties:

- *Data-specific*: AE is only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific for the given training data, they are different than a standard data compression algorithm like gzip
- *Lossy*: The output of the AE will not be exactly the same as the input, it will be a close but degraded representation.
- *Unsupervised*: AE don't need explicit labels to train on

## 2.1 Auto-Encoder Architecture

Traditionally, AE is one of the essential algorithms for dimensionality reduction (e.g Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE))

AE consists of 2 functions: encoder function ($g_\phi$) parameterized by $\phi$ and decoder function ($f_\theta$) parameterized by $\theta$. The low-dimensional code learned for input $x$ in the bottleneck layer is $z = g_\phi(\mathbf{x})$ and the reconstructed input is $\mathbf{x}' = f_\theta\left(g_\phi(\mathbf{x})\right)$ [Wen18]

Fig. 3 shows illustration of AE model architecture

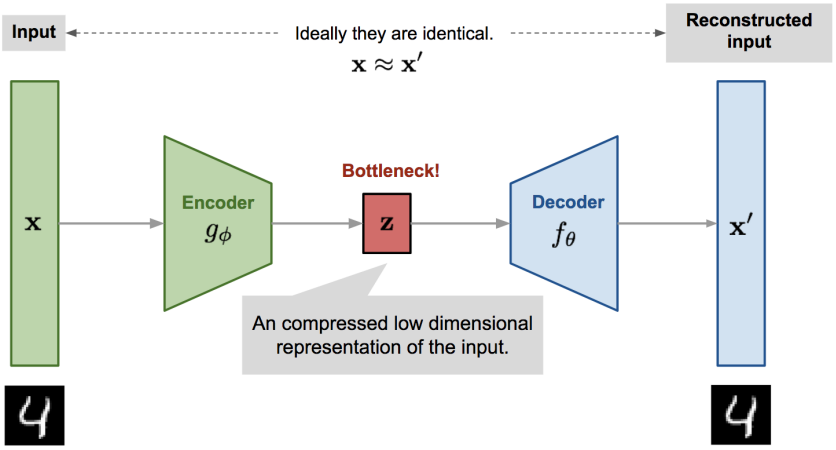


Figure 3: Illustration of autoencoder model architecture [Wen18]

*Encoder network*: It translates the original high-dimension input into the latent low-dimensional code. The input size is larger than the output size.

*Decoder network*: The decoder network recovers the data from the code, likely with larger and larger output layers.

The parameters $(\theta, \phi)$ are learned together to output a reconstructed data sample same as the original input, $\mathbf{x} \approx f_\theta\left(g_\phi(\mathbf{x})\right)$, or in other words, to learn an identity function. There are various metrics to quantify the difference between two vectors, such as cross entropy when the activation function is sigmoid, or as simple as Mean Squared Error (MSE) loss:

$$L_{\mathrm{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \left( \mathbf{x}^{(i)} - f_\theta \left( g_\phi \left( \mathbf{x}^{(i)} \right) \right) \right)^2 \tag{19}$$

where $\mathbf{x}^{(i)}$ is each of data point and is a vector of $d$ dimensions, $\mathbf{x}^{(i)} = \left[ x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)} \right]$

## 2.2 Auto-Encoder for Financial Feature Learning

DL constructs a Deep Neural Network (DNN) to hierarchically transform the information from layer to layer. Such deep transformation encourages much informative feature representations for a specific learning task. AE is adopted to initialize the deep transformation part after the fuzzy transformation. [YDD16]

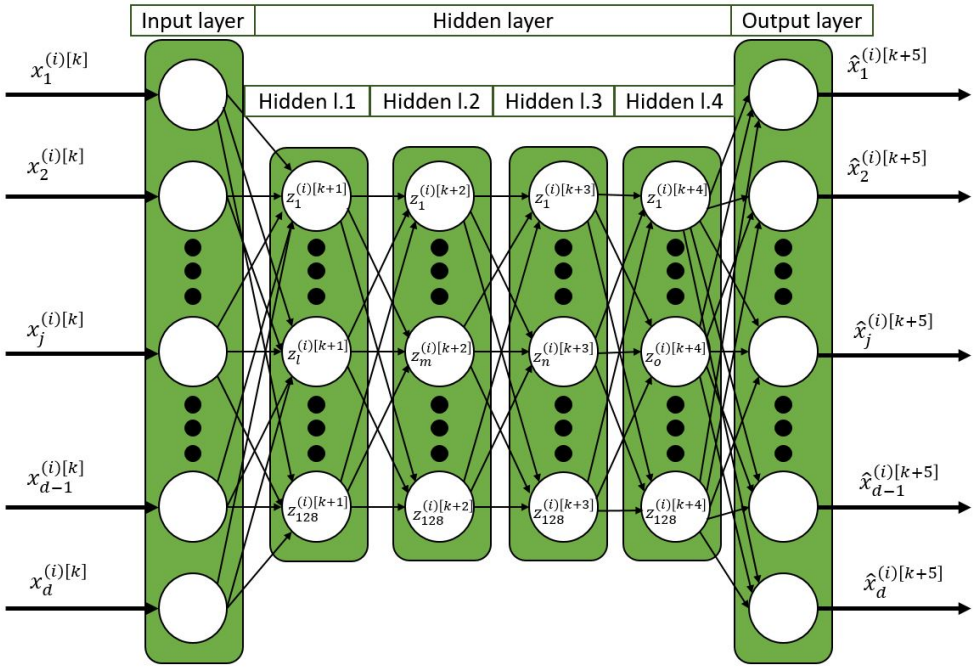


Figure 4: Illustration of autoencoder model for deep transformation

Fig. 4 shows illustration of AE initialization for deep transformation. We define value of the weight for $j^{th}$ hidden unit in $i^{th}$ training example in layer $k$ ($w_j^{(i)[k]}$), value of the bias for $j^{th}$ hidden unit in $i^{th}$ training example in layer $k$ ($b_j^{(i)[k]}$),value of the feature $j$ in $i^{th}$ training example in layer $k$ ($x_j^{(i)[k]}$), value of the feature $l$ in $i^{th}$ training example in layer $k+1$ ($z_l^{(i)[k+1]}$), value of the feature $m$ in $i^{th}$ training example in layer $k+2$ ($z_m^{(i)[k+2]}$), value of the feature $n$ in $i^{th}$ training example in layer $k+3$ ($z_n^{(i)[k+3]}$), value of the feature $o$ in $i^{th}$ training example in layer $k+4$ ($z_o^{(i)[k+4]}$) and predicted value $j$ from $i^{th}$ training example in layer $k+5$ ($\hat{x}_j^{(i)[k+5]}$). The number of hidden layer is 4 and the node number per hidden layer is fixed to 128. [YDD16]

Functions of the nodes in each of the 6 layers are defined as following:

- *Input layer:* The nodes transmit the input values from fuzzy output (18) to the next layer directly with the link weight $\mathbf{w^{(i)[k]}} \in R^{d \times 1}$ and bias are unity.

$$\mathbf{x^{(i)[k]}} = \mathbf{o^{(l)}} \tag{20}$$

  where $d = m \cdot 3$, since three clusters are chosen.

- *Hidden layer 1:* Compressing the input from input layer into 128 hidden unit and apply sigmoid function ($\sigma$) as activation function with the link weight $\mathbf{w^{(i)[k+1]}} \in R^{d \times 128}$ and bias weight $\mathbf{b^{(i)[k+1]}} \in R^{128 \times 1}$ are randomly initialized.

$$\mathbf{z^{(i)[k+1]}} = \sigma(\mathbf{w^{(i)[k+1]}}^T \mathbf{x^{(i)[k]}} + \mathbf{b^{(i)[k+1]}}) \tag{21}$$

- *Hidden layer 2, 3 & 4:* Linear transformation the output from *Hidden layer 1, 2 and 3 respectively* into 128 hidden unit and apply $\sigma$ as activation function with the link weight $\mathbf{w^{(i)[k+2]}}, \mathbf{w^{(i)[k+3]}}, \mathbf{w^{(i)[k+4]}} \in R^{128 \times 128}$ and bias weight $\mathbf{b^{(i)[k+2]}}, \mathbf{b^{(i)[k+3]}}, \mathbf{b^{(i)[k+4]}} \in R^{128 \times 1}$ are randomly initialized.

$$\mathbf{z^{(i)[k+2]}} = \sigma(\mathbf{w^{(i)[k+2]}}^T \mathbf{z^{(i)[k+1]}} + \mathbf{b^{(i)[k+2]}}) \tag{22}$$

$$\mathbf{z^{(i)[k+3]}} = \sigma(\mathbf{w^{(i)[k+3]}}^T \mathbf{z^{(i)[k+2]}} + \mathbf{b^{(i)[k+3]}}) \tag{23}$$

$$\mathbf{z^{(i)[k+4]}} = \sigma(\mathbf{w^{(i)[k+4]}}^T \mathbf{z^{(i)[k+3]}} + \mathbf{b^{(i)[k+4]}}) \tag{24}$$

- *Output layer:* Decompressing the output from Hidden layer 4 into $d$ units with the link weight $\mathbf{w^{(i)[k+5]}} \in R^{128 \times d}$ and bias are unity so as to make it the same size with the input layer, after that the layer is applied $\sigma$ activation function.

$$\mathbf{\hat{x}^{(i)[k+5]}} = \sigma(\mathbf{w^{(i)[k+5]}}^T \mathbf{z^{(i)[k+4]}} + \mathbf{b^{(i)[k+5]}}) \tag{25}$$

For ease of notation, we define above 6 steps into 2 steps

- *Encode function:*

$$\mathbf{z} = g_\phi(\mathbf{o^{(l)}}) \tag{26}$$

- *Decode function:*

$$\mathbf{\hat{x}^{(i)[k+5]}} = f_\theta(\mathbf{z}) \tag{27}$$

The AE is optimized with following loss

$$\sum_t \left\| \mathbf{x^{(i)[k]}} - f_\theta \left( g_\phi \left( \mathbf{\hat{x}^{(i)[k+5]}} \right) \right) \right\|_2^2 + \eta \left\| \mathbf{w}^{(j)} \right\|_2^2 \tag{28}$$

where $j = (k+1), ..., (k+4)$

# 3 Direct Reinforcement

RL is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal over time. RL uses the formal framework of Markov Decision Processes (MDP) to define the interactions between a learning agent and its environment in terms of states, actions, and rewars. [SB18].

According to different learning objectives, RL algorithms can be classified as either Direct Reinforcement (DR) (sometimes called "policy search"), *Value Function* or *Actor-Critic* methods.

*DR* refers to algorithms that *do not* have to learn a value function in order to derive a policy. [MS01]

For many financial decision making problem, however, results accrue gradually over time, and one can immediately measure short-term performance. This enables use of a DR approach to provide immediate feedback to optimize the strategy.

## 3.1 Structure of Trading Systems

We consider agents that trade fixed position sizes in a single security. Here, our traders are assumed to take only long, neutral and short position, $F_t \in \{1, 0, -1\}$, of constant magnitude. The price series being traded is denoted $z_t$

A single asset trading system that takes into account transaction costs and market impact has the following decision function

$$\begin{aligned} F_t &= F\left(\theta_t; F_{t-1}, I_t\right) \quad \text{with} \\ I_t &= \{z_t, z_{t-1}, z_{t-2}, \ldots; y_t, y_{t-1}, y_{t-2}, \ldots\} \end{aligned} \tag{29}$$

where $\theta_t$ denotes the (learned) system parameters at time $t$ and $I_t$ denotes the information set at time $t$, which includes present and past values of the price series $z_t$ and an arbitrary number of other external variables denoted $y_t$ A single example is a long, short trader with $m+1$ autoregressive inputs

$$F_t = \text{sign}\left(uF_{t-1} + v_0 r_t + v_1 r_{t-1} + \ldots + v_m r_{t-m} + w\right) \tag{30}$$

where $r_t$ are the price returns of $z_t$ (defined below) and the system parameters $\theta$ are the weights $\{u, v_i, w\}$ The above equation (30) describes a discrete-action, deterministic trader. When discrete values $F_t \in \{1, 0, -1\}$ are imposed, however, the decision function is not differentiable, so we replace *sign* with *tanh* during learning and discretizing the outputs when trading.

## 3.2 Performance Criteria

In general, the performance criteria that we consider are functions of profit and wealth $U(W_T)$ after a sequence of $T$ time steps, or more generally of the whole time sequence of trades

$$U\left(W_T, \ldots, W_t, \ldots, W_1, W_0\right) \tag{31}$$

The simple form $U(W_T)$ includes standard economic utility functions. The second case is the general form for path-dependent performance functions, which include inter-temporal utility functions and performance ratios like the Sharpe ratio and Sterling ratio. In either case, performance criterion at time $T$ can be expressed as a function of the sequence of trading returns

$$U\left(R_T, \ldots, R_t, \ldots, R_2, R_1; W_0\right) \tag{32}$$

For brevity, we denote this general form by $U_T$

## 3.3 Profit and Wealth for Trading Systems

Trading systems can be optimized by maximizing performance functions, $U()$, such as profit, wealth, utility functions of wealth or performance ratios like the Sharp ratio.

*Additive profits* are to consider if each trade is for a fixed number of shares or contracts of security $p_t$. We define $r_t = p_t - p_{t-1}$ and $r_t^f = p_t^f - p_{t-1}^f$ as *price returns* of a risky (traded) asset and a risk-free asset respectively. The additive profit accumulated over $T$ time periods with trading position size $\zeta$ is defined in term of the *trading returns*, $R_t$, as:

$$\begin{aligned} P_T &= \sum_{t=1}^{T} R_t \quad \text{where} \\ R_t &\equiv \zeta \left\{ r_t^f + F_{t-1} \left( r_t - r_t^f \right) - \delta \left| F_t - F_{t-1} \right| \right\} \end{aligned} \tag{33}$$

with initial condition $P_0 = 0$ and $F_T = F_0 = 0$. If risk-free rate of interest is ignored ($r_t^f = 0$), a simplified expression is obtained:

$$R_t = \zeta \left\{ F_{t-1} r_t - \delta \left| F_t - F_{t-1} \right| \right\} \tag{34}$$

The wealth of the trader is defined as $W_T = W_0 + P_T$

*Multiplicative profits* are appropriate when a fixed fraction of accumulated wealth $v > 0$ is invested in each long or short trade. Here, $r_t = \left( \frac{p_t}{p_{t-1}} - 1 \right)$ and $r_t^f = \left( \frac{p_t^f}{p_{t-1}^f} - 1 \right)$. If no short sales are allowed and the leverage factor is set fixed at $v = 1$, the wealth at time $T$ is:

$$W_T = W_0 \prod_{t=1}^{T} \left\{ 1 + R_t \right\} \text{ where } \quad \left\{ 1 + R_t \right\} \equiv \left\{ 1 + (1 - F_{t-1}) r_t^f + F_{t-1} r_t \right\} \times \left\{ 1 - \delta \left| F_t - F_{t-1} \right| \right\} \tag{35}$$

If risk-free rate of interest is ignored ($r_t^f = 0$), a second simplified expression is obtained:

$$\left\{ 1 + R_t \right\} = \left\{ 1 + F_{t-1} r_t \right\} \left\{ 1 - \delta \left| F_t - F_{t-1} \right| \right\} \tag{36}$$

### 3.4 Recurrent Reinforcement Learning

Given a trading system model $F_t(\theta)$, the goal is to adjust $\theta$ in order to maximize $U_T$. For traders of form (29) and trading returns of form (34) or (36), the gradient of $U_T$ with respect to the parameters $\theta$ of the system after a sequence of $T$ periods is:

$$\frac{dU_T(\theta)}{d\theta} = \sum_{t=1}^{T} \frac{dU_T}{dR_t} \left\{ \frac{dR_t}{dF_t} \frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}} \frac{dF_{t-1}}{d\theta} \right\} \tag{37}$$

The parameters $\theta$ of the maximization problem can be optimized using gradient ascent. (with learning rate $\rho$)

$$\Delta\theta = \rho \frac{dU_T(\theta)}{d\theta} \tag{38}$$

Note that due to the inherent recurrence, the quantities $dF_t/d\theta$ are *total derivatives* that depend upon the entire sequence of previous time periods. To correctly compute and optimize these total derivatives in an efficient manner requires an approach similar to Back-Propagation Through Time (BPTT) [RHW86, Wer90]. The temporal dependencies in a sequence of decisions are accounted for through a recursive update equation for the parameter gradients:

$$\frac{dF_t}{d\theta} = \frac{\partial F_t}{\partial \theta} + \frac{\partial F_t}{\partial F_{t-1}} \frac{dF_{t-1}}{d\theta} \tag{39}$$

The equation (37) (39) are assumed differentiability of $F_t$.

### 3.5 Value Functions and Q-Learning

Besides explicitly training to take actions, we can also implicit learn to correct actions through the technique of *value iteration*. Value iteration uses a *value function* to evaluate and

improve policies. The value function, $V^{\pi}(x)$, is an estimate of discounted future rewards that will be received from the starting state $x$, and by following the policy $\pi$ there after.

The value function satisfies *Bellman's equation*

$$V^{\pi}(x) = \sum_{a} \pi(x,a) \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V^{\pi}(y) \right\} \tag{40}$$

where $\pi(x,a)$ is the probability of taking action $a$ in state $x, p_{xy}(a)$ is the probability of transitioning from state $x$ to state $y$ when taking action $a, D(x,y,a)$ is the immediate reward from taking action $a$ and transitioning from state $x$ to state $y$ and $\gamma$ is the discount factor that weighs the importance of future rewards versus immediate rewards.

A policy is an *optimal* policy if its value function is greater than or equal to the value of all other policies for a given set of states. The optimal value function is defined as:

$$V^{*}(x) = \max_{\pi} V^{\pi}(x) \tag{41}$$

and satisfies the Bellman's optimality equations

$$V^{*}(x) = \max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V^{*}(y) \right\} \tag{42}$$

is guaranteed to converge to the optimal value function under certain general conditions. The optimal policy can be determined from the optimal value function through.

$$a^{*} = \arg\max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V^{*}(y) \right\} \tag{43}$$

### 3.6 Q-Learning

Q-Learning estimate future rewards based on both the current state and the current action taken [Wat89]. We can write the Q-function version of Bellman's optimality equation as:

$$Q^{*}(x,a) = \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma \max_{b} Q^{*}(y,b) \right\} \tag{44}$$

Q-function can be learned using a value iteration approach:

$$Q_{t+1}(x,a) = \max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma Q_{t}(y) \right\} \tag{45}$$

This has been shown to converge to the optimal Q-function, $Q^{*}(x,a$ [Wat89]. One calculates the best action as:

$$a^{*} = \arg\max_{a} \left( Q^{*}(x,a) \right) \tag{46}$$

The update rule for training a function approximator is then based on the gradient of the error.

$$\frac{1}{2} \left( D(x,y,a) + \gamma \max_{b} Q(y,b) - Q(x,a) \right)^{2} \tag{47}$$

### 3.7 Advantage Updating

Advantage Updating is a refinement of Q-Learning algorithm [IP15]. Advantage Updating was developed to deal with continous-time RL problems, though it is applicable to the discrete-time as well. Also, Advantage Learning has been shown to be able to learn at a much faster rate than Q-Learning in the presence of noise.

Advantage Updating learns two separate functions: the advantage function $A(x,a)$ and the value function $V(x)$. The advantage function measures the relative change in the value of choosing action $a$ while in state $x$ versus choosing the best possible action for that state. The value function measures the discounted future reward as described previously. Advantage Updating has the following relationship with Q-Learning:

$$Q^{*}(x,a) = V^{*}(x) + A^{*}(x,a) \tag{48}$$

Similarly to Q-Learning, the optimal action to take in state $x$ is found by $a^{*} = \arg\max_{a} \left( A^{*}(x,a) \right)$.

### 3.8 Deep Q-Network

Deep Q-Network was proposed by Mnih et al, 2013 [MKS+13]. Considering tasks in which an agent interacts with an environment $\varepsilon$ in a sequence of actions, observations $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$ and rewards $r_t$. At each time-step, the agent selects an action $a_t$ from the set of legal game actions, $A = \{1, ..., K\}$. All sequences are assumed to terminate in a finite number of time steps and future rewards are discounted by a factor of $\gamma$ per time-step. Consequently, future discounted return at time $t$ is defined as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where $T$ is the terminal time step. Maximum expected return achievable by following any strategy, after seeing some sequence $s$ and taking some action $a$ is defined as optimal action-value function $Q^*(s, a) = \max_\pi \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]$, where $\pi$ is a policy mapping sequences to actions (or distributions over action). Following the famous *Bellman equation*, $Q^*(s, a)$ are defined as:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^* \left(s', a'\right) | s, a \right] \tag{49}$$

In practice, $Q^*(s, a)$ may be impossible when the action or state is finite, without any generalization. Instead, it is common to use a function approximation $Q(a, s; \theta) \approx Q^*(s, a)$. By using a non-linear function approximator such as Q-Network with weight $\theta$, which refer as *Deep Q-Network* ,we can approximate $Q^*(s, a)$ by minimising a sequence of loss function $L_i(\theta_i)$ that changes at each iteration $i$.

$$L_i\left(\theta_i\right) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[ \left(y_i - Q\left(s, a; \theta_i\right)\right)^2 \right] \tag{50}$$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) | s, a \right]$ is the target for iteration $i$ and $\rho(s, a)$ is a probability distribution over sequences $s$ and actions $a$. Optimization of weight $\theta$ is updated by differentiation the loss function with respect to the weights $\theta$.

$$\nabla_{\theta_i} L_i\left(\theta_i\right) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ \left( r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) - Q\left(s, a; \theta_i\right) \right) \nabla_{\theta_i} Q\left(s, a; \theta_i\right) \right] \tag{51}$$

# 4 Methodology

## 4.1 Unified Trading Framework

Based on the equation (29), we extend the feature vector for the unified trading framework.

$$\mathbf{f_t} = [d_{t-m+1}, ..., d_t, ..., d_{kt}, s_{t-m+1}, ..., s_t, e_{t-m+1}, ..., e_t, n_t] \tag{52}$$

whereas $\mathbf{f_t}$ is a combination of the four vector:

- *Fuzzied return vector* $\mathbf{d} = d_{t-m+1}, ..., d_t$: defined as the fuzzy representation vector from k cluster obtained from recent m returns values, which is determined by $d_t = v_i(z_t)$ and $z_t = p_t - p_{t-1}$, following the definition from equation (18).
- *Sentiment vector* $\mathbf{s} = s_{t-m+1}, ..., s_t$: defined as the sentimental reaction of trader from news. Section 4.1.1 reviews the sentimental analysis method and explains how news are scraped.
- *Earning per share vector* $\mathbf{e} = e_{t-m+1}, ..., e_t$: defined as the reported Earning per share (EPS), calculated from the Quarterly Financial Statement (QFS). Section 4.1.2 describes the method used for the calculation.
- *Stock news vector* $\mathbf{n} = n_t$: defined as embedded vector of all news in day $t$. Section 4.1.3 introduces the detailed implementation of the embedding method.

Moreover, we consider the method to concatenate the *four vectors* from *Close price* of Fuzzied return vector and *published date* of the other vectors. Section 4.1.4 introduces the method we applied for the concatenation.

After obtaining the feature vector $f_t$, while admitting the general effectiveness of Fuzzy Deep Recurrent Neural Network (FDRNN) framework, we keep the rest as is in order to verify the effectiveness of the extended feature vector.

$$\begin{aligned} &\max_{\{\Theta, g_d(\cdot), v(\cdot)\}} U_T(R_1..R_T) \\ &\text{s.t. } R_t = \delta_{t-1} z_t - c\,|\delta_t - \delta_{t-1}| \\ &\delta_t = \tanh(\langle \mathbf{w}, \mathbf{F}_t \rangle + b + u\delta_{t-1}) \\ &\mathbf{F}_t = g_d(v(\mathbf{f}_t)) \end{aligned} \tag{53}$$

### 4.1.1 Sentimental Reaction

As the internet becomes popular, the use of internet for financial analysis, fundamental analysis and technical analysis has been increasing sharply. Financial news and event are one of the most important factor to be considered for people' reaction to an increasing/decreasing trend of stock, affected by positive, negative and neutral image created from them. Assuming a perfect market, we collected Vinamilk (VNM) financial news and events between 2009 and 2020 for the purpose of testing for the unified trading framework. We crawled the news from Cafef.vn website [1]. We prepared the text in these VNM financial news and events as follows:

1. We create and scrape a *vinamilk-news* spider [2] for the purpose of obtaining the *publish date, url and title* of the VNM financial news and events.
2. We extract text from Hypertext Markup Language (HTML) tags [3] with *ID news-content*.
3. We translate the text from Vietnamese version to English version using Google Translate API [4] in order to reuse the sentimental corpus of the English language. We consider the

[1] https://s.cafef.vn/tin-doanh-nghiep/vnm/Event.chn
[2] https://github.com/scrapy/scrapy
[3] https://code.launchpad.net/ leonardr/beautifulsoup/bs4
[4] https://github.com/ssut/py-googletrans

fact that as of 2020, research of NLP for English language reaches state-of-the-art with the use of generative models (i.e GPT) [Rad18] and models for language understanding (i.e BERT) [DCLT18]

Following table descries an example of the collected text

| Column | Example |
|---|---|
| Publish date | 20/02/2020 14:35 |
| Title | Vinamilk đủ điều kiện xuất khẩu một số sản phẩm sữa vào thị trường Trung Quốc |
| Vietnamese text | Vụ Thị trường châu Á - châu Phi (Bộ Công Thương) cho biết, Tổng cục Hải quan Trung Quốc vừa chính thức cấp mã giao dịch và cho phép 1 nhà máy của Công ty Cổ phần Sữa Việt Nam (Vinamilk) được phép xuất khẩu sản phẩm sữa đặc có đường (sweetened condensed milk) và các loại sữa đặc khác (other condensed milk) vào thị trường Trung Quốc.Tổng cục Hải quan Trung Quốc cũng cho biết cơ quan này đang tiếp tục xem xét, đánh giá hồ sơ, tài liệu của các nhà máy và công ty sữa khác của Việt Nam.Đây là tin vui cho ngành sữa Việt Nam trong đầu năm 2020, là kết quả của những nỗ lực của các cơ quan quản lý trong việc đồng hành cùng các doanh nghiệp trong thời gian qua.Tuy nhiên,theo ý kiến của một số chuyên gia thì sữa đặc có đường và các loại sữa đặc khác chưa phải là sản phẩm chủ lực của Vinamilk, doanh nghiệp chế biến sữa lớn nhất Việt Nam. Điều này cho thấy Bộ Nông nghiệp và PTNT vẫn còn nhiều việc phải làm để giúp ngành sữa Việt Nam có được giấy thông hành toàn diện khi đi vào thị trường Trung Quốc, qua đó tận dụng được mức thuế nhập khẩu ưu đãi mà Việt Nam đã đàm phán được với Trung Quốc từ hơn 10 năm trước.Sữa Việt Nam sẽ xuất khẩu chính ngạch sang Trung Quốc Quỳnh Anh Theo Nhịp sống kinh tế |
| English text | Market Department Asia - Africa (MOIT) said the General Administration of Customs of China has officially granted the transaction and allow 1 plant JSC Vietnam Dairy Products (Vinamilk) is allowed to export products sweetened condensed milk (sweetened condensed milk) and other kinds of milk (condensed milk other) into the Chinese market Quoc.Tong China Customs said the agency is continuing to consider and evaluate records documents of factories and other dairy companies of Vietnam Nam.Day is good news for the dairy sector in Vietnam in early 2020, is the result of efforts by the agency to accompany the letter during qua.Tuy course now, in the opinion of some experts, condensed milk and other kinds of milk is not the flagship product c ủa Vinamilk, dairy processing enterprises in Vietnam. This shows that the Ministry of Agriculture and Rural Development is still much to do to help the dairy sector in Vietnam has been laissez comprehensive entering the Chinese market, which take advantage of the import tariff incentives that Vietnam was negotiating with China for over 10 years truoc.Sua Vietnam will export quota to the Middle Quoc Quynh Anh Theo economic Lifestyle |

**Sample of collected VNM financial news and events**

We create sentimental feature, hereby call as *sentimental reaction*, by using NLP techniques for sentimental analysis. There has been a vast amount of study for sentimental analysis, among them, RNN and LSTM [HS97] has been firmly established as baseline approach for sequence modeling in general and sentimental analysis in particular. Attention mechanism has become an important development in order to allow sequence modeling of dependencies without regard to their distance in the input or output sequences [BCB14]. Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations. *The Transformer* is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNN or convolution [VSP+17]. For ease of implementation, we introduce *HuggingFace's Transformers library*, a library for state-of-the-art NLP, making these developments available to the community by gathering state-of-the-art general-purpose pretrained models under a unified API together with an ecosystem of libraries [WDS+19]. We apply pretrained-model DistilBERT [SDCW19], a distilled version of BERT [DCLT18] to create the *sentimental reaction*.

### 4.1.2 Fundamental EPS

Traditionally, QFS are published within 15-30 days after the end of the financial quarter in Vietnamese market. Their formats are mostly published in Portable Document Format (PDF) and in fact there was any research to extract these report automatically. Therefore, to create a feature from fundamental analysis, we consider to manually calculate the quarter EPS based on QFS. Base on the VNM stocks choice from 4.1.1, we collected and prepared the VNM *fundamental EPS* feature between 2009 and 2020 as follows

1. We download the VNM QFS from Cafef.vn website [5]
2. We obtain and fill in information about *publish date*, *profits after enterprise income tax* and *number of outstanding shares*
3. We calculate VNM *fundamental EPS* with the equation:

$$EPS = \frac{\text{Profits after enterprise income tax}}{\text{Number of outstanding shares}} \tag{54}$$

### 4.1.3 Embedded News

For the embedding we used the Vietnamese text feature from section 4.1.1, we consider not to reuse the sentimental corpus from English language, we can create a private Vietnamese corpus based on the collected news so far.

*Tf-idf* stands for term frequency-inverse document frequency, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. Therefore, for ease of simplicity, we use tf-idf as a baseline method to see whether the agent can make prediction well or not.

While we achieved our embedding news vector, raw text features can be very sparse. In an attempt to address this potential sparsity, we incorporated dimensionality reduction in our feature, PCA are used to incorporate with the problem.

### 4.1.4 Overlapping Processing

On the assumption that news will effect the price of the next close, we update the *publish date* feature of news and EPS to the date of the price of the previous close with following

[5] http://s.cafef.vn/hose/VNM-cong-ty-co-phan-sua-viet-nam.chn

algorithm.

**Result:** The date of the price of previous close
List of date of the price;
A particular publish date to update;
**while** *List not end* **do**
    Get elem in list;
    Compute the difference between elem and publish date;
    **if** *difference is* $\geq$ *0* **then**
        Store the elem as minimum;
        Compare the minimum with previous minimum, get the smallest;
    **end**
    Return the minimum elem;
**end**

**Algorithm 1:** Overlapping algorithms

After overlaps the date of the price of previous close to the publish date of the other three vectors, we concatenate the features $f_t$ by using *left-join*

### 4.2 Data

The data included in the optimization procedure are daily series on stock price. The returns are computed for VNM listed on Vietnam stock marke. The price data begins from $31^{th}$ December 2009 to $21^{th}$ April 2020.

### 4.3 Performance evaluation methodology

To test the effectiveness of the agent, we consider the accuracy of the agent at each time the agent make an action, whether the action is correct or not, in short, the accuracy of the long and short action

### 4.4 Transaction costs

Transaction cost were pointed at 0.001% and changed to 0.0025% and 0.005% to verify the effect of the reward function. Therefore, considering the inclusion of transaction costs and incorporating it into the model will help the investors to control the amount of trading and rebalancing.

### 4.5 Performance measures

Optimized portfolio's performance is measured along with the annually return, volatility(standard validation), Sharpe ratio and maximum drawn-down after backtesting procedure.

## 5 Empirical Results

## 6 Conclusion

# A References

# References


[Bal87] Dana H. Ballard. Modular learning in neural networks. In *Proceedings of the sixth National conference on Artificial intelligence*, 1987.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[BR78] M. Braae and D.A. Rutherford. Fuzzy relations in a control setting. *Kybernetes*, 7(3), 1978.

[CHLLC06] Alan Liu Chiung-Hon Leon Lee and Wen-Sung Chen. Pattern discovery of fuzzy time series for financial prediction. *IEEE Transactions on Knowledge and Data Engineering*, 18(5), 2006.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 10 2018.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[IP15] Leemon C. Baird III and Bill Parks. Exhaustive attack analysis of BBC with glowworm for unkeyed jam resistance. In Qinqing Zhang, Jerry Brand, Thomas G. MacDonald, Bharat T. Doshi, and Bonnie L. Gorsic, editors, *34th IEEE Military Communications Conference, MILCOM 2015, Tampa, FL, USA, October 26-28, 2015*, pages 300–305. IEEE, 2015.

[KF88] George J. Klir and Tina A. Folger. *Fuzzy Sets, Uncertainty, and Information.* Prentice Hall, 1988.

[LL91] Chin-Teng Lin and C. S. George Lee. Neural-network-based fuzzy logic control and decision system. *IEEE Transaction On Computers*, 40(12), 1991.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 12 2013.

[MS01] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 12:875–89, 07 2001.

[PB94] N.R. Pal and J.C. Bezdek. Measuring fuzzy uncertainty. *IEEE Transactions on Fuzzy Systems*, 2:107–118, 1994.

[Rad18] Alec Radford. Improving language understanding by generative pre-training, 2018.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, chapter 8, pages 318–362. MIT Press, Cambridge, MA, 1986.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[Sch15] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[SDCW19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 10 2019.

[SROASP19] Marc Sanchez-Roger, María Dolores Oliver-Alfonso, and Carlos Sanchís-Pedregosa. Fuzzy logic and its uses in finance: A systematic review exploring its potential to deal with banking crises. In *Fuzzy Sets, Fuzzy Logic and Their Applications*, 2019.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[Wat89] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

[WDS+19] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

[Wen18] Lilian Weng. From autoencoder to beta-vae. *lilianweng.github.io/lil-log*, 2018.

[Wer90] P. Werbos. Backpropagation through time: what does it do and how to do it. In *Proceedings of IEEE*, volume 78, pages 1550–1560, 1990.

[YDD16] Youyong Kong Zhiquan Ren Yue Deng, Feng Bao and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28:653–664, 2016.