Deep Direct Reinforcement Learning for Financial Signal Representation and Trading

Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai, *Senior Member, IEEE*

***Abstract*— Can we train the computer to beat experienced traders for financial assert trading? In this paper, we try to address this challenge by introducing a recurrent deep neural network (NN) for real-time financial signal representation and trading. Our model is inspired by two biological-related learning concepts of deep learning (DL) and reinforcement learning (RL). In the framework, the DL part automatically senses the dynamic market condition for informative feature learning. Then, the RL module interacts with deep representations and makes trading decisions to accumulate the ultimate rewards in an unknown environment. The learning system is implemented in a complex NN that exhibits both the deep and recurrent structures. Hence, we propose a task-aware backpropagation through time method to cope with the gradient vanishing issue in deep training. The robustness of the neural system is verified on both the stock and the commodity future markets under broad testing conditions.**

***Index Terms*— Deep learning (DL), financial signal processing, neural network (NN) for finance, reinforcement learning (RL).**

NOMENCLATURE

AE	Autoencoder.
BPTT	Backpropagation through time.
DL	Deep learning.
DNN	Deep neural network.
DRL	Direct reinforcement learning.
DDR	Deep direct reinforcement.
FDDR	Fuzzy deep direct reinforcement.
RDNN	Recurrent DNN.
RL	Reinforcement learning.
NN	Neural network.
SR	Sharpe ratio.
TP	Total profits.

I. INTRODUCTION

TRAINING intelligent agents for automated financial asserts trading is a time-honored topic that has been widely discussed in the modern artificial intelligence [1]. Essentially, the process of trading is well depicted as an online decision making problem involving two critical steps of market condition summarization and optimal action execution. Compared with conventional learning tasks, dynamic decision making is more challenging due to the lack of the supervised information from human experts. It, thus, requires the agent to explore an unknown environment all by itself and to simultaneously make correct decisions in an online manner.

Such self-learning pursuits have encouraged the long-term developments of RL—a biological inspired framework—with its theory deeply rooted in the neuroscientific field for behavior control [2]–[4]. From the theoretical point of view, stochastic optimal control problems were well formulated in a pioneering work [2]. In practical applications, the successes of RL have been extensively demonstrated in a number of tasks, including robots navigation [5], atari game playing [6], and helicopter control [7]. Under some tests, RL even outperforms human experts in conducting optimal control policies [6], [8]. Hence, it leads to an interesting question in the context of trading: can we train an RL model to beat experienced human traders on the financial markets? When compared with conventional RL tasks, algorithmic trading is much more difficult due to the following two challenges.

The first challenge stems from the difficulties in financial environment summarization and representation. The financial data contain a large amount of noise, jump, and movement leading to the highly nonstationary time series. To mitigate data noise and uncertainty, handcraft financial features, e.g., moving average or stochastic technical indicators [9], are usually extracted to summarize the market conditions. The search for ideal indicators for technical analysis [10] has been extensively studied in quantitative finance. However, a widely known drawback of technical analysis is its poor generalization ability. For instance, the moving average feature is good enough to describe the trend but may suffer significant losses in the mean-reversion market [11]. Rather than exploiting predefined handcraft features, can we learn more robust feature representations directly from data?

The second challenge is due to the dynamic behavior of trading action execution. Placing trading orders is a systematic work that should take a number of practical factors into consideration. Frequently changing the trading positions (long or short) will contribute nothing to the profits but lead to great losses due to the transaction cost (TC) and slippage. Accordingly, in addition to the current market condition, the historic actions and the corresponding positions are, meanwhile, required to be explicitly modeled in the policy learning part.

Manuscript received April 30, 2015; revised October 21, 2015 and January 22, 2016; accepted January 22, 2016. Date of publication February 15, 2016; date of current version February 15, 2017. This work was supported by the Project of the National Natural Science Foundation of China under Grant 61327902 and Grant 61120106003. The work of Y. Kong was supported by National Science Foundation of Jiangsu Province, China, under Grant BK20150650.

Y. Deng is with the Automation Department, Tsinghua University, Beijing 100084, China, and also with the School of Pharmacy, University of California at San Francisco, San Francisco, CA 94158 USA (e-mail: yuedeng.thu@gmail.com).

F. Bao, Z. Ren, and Q. Dai are with the Automation Department, Tsinghua University, Beijing 100084, China (e-mail: fbao0110@gmail.com; renzhiquan1989@gmail.com; qhdai@tsinghua.edu.cn).

Y. Kong is with the School of Computer Science and Engineering, Southeast University, Nanjing 210000, China (e-mail: kongyouyong@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2016.2522401

Without adding extra complexities, how can we incorporate such memory phenomena into the trading system?

In addressing the aforementioned two questions, in this paper, we introduce a novel RDNN structure for simultaneous environment sensing and recurrent decision making for online financial assert trading. The bulk of the RDNN is composed of two parts of DNN for feature learning and recurrent neural network (RNN) for RL. To further improve the robustness for market summarization, the fuzzy learning concepts are introduced to reduce the uncertainty of the input data. While the DL has shown great promises in many signal processing problems as image and speech recognitions, to the best of our knowledge, this is the first paper to implement DL in designing a real trading system for financial signal representation and self-taught reinforcement trading.

The whole learning model leads to a highly complicated NN that involves both the deep and recurrent structures. To handle the recurrent structure, the BPTT method is exploited to unfold the RNN as a series of time-dependent stacks without feedback. When propagating the RL score back to all the layers, the gradient vanishing issue is inevitably involved in the training phase. This is because the unfolded NN exhibits extremely deep structures on both the feature learning and time expansion parts. Hence, we introduce a more reasonable training method called the task-aware BPTT to overcome this pitfall. In our approach, some virtual links from the objective function are directly connected with the deep layers during the backpropagation (BP) training. This strategy provides the deep part a chance to see what is going on in the final objective and, thus, improves the learning efficiency.

The DDR trading system is tested on the real financial market for future contracts trading. In detail, we accumulate the historic prices of both the stock-index future (IF) and commodity futures. These real market data will be directly used for performance verifications. The deep RL system will be compared with other trading systems under diverse testing conditions. The comparisons show that the DDR system and its fuzzy extension are much robust to different market conditions and could make reliable profits on various future markets.

The remaining parts of this paper are organized as follows. Section II generally reviews some related works about the RL and the DL. Section III introduces the detailed implementations of the RDNN trading model and its fuzzy extension. The proposed task-aware BPTT algorithm will be presented in Section IV for RDNN training. Section V is the experimental part where we will verify the performances of the DDR and compare it with other trading systems. Section VI concludes this paper and indicates some future directions.

II. Related Works

RL [12] is a prevalent self-taught learning [13] paradigm that has been developed to solve the Markov decision problem [14]. According to different learning objectives, typical RL can be generally categorized into two types as critic-based (learning value functions) and actor-based (learning actions) methods. Critic-based algorithms directly estimate the value functions that are perhaps the mostly used RL frameworks in the filed. These value-function-based methods, e.g., TD-learning or Q-learning [15] are always applied to solve the optimization problems defined in a discrete space. The optimizations of value functions can always be solved by dynamic programming [16].

While the value-function-based methods (also known as critic-based method) perform well for a number of problems, it is not a good paradigm for the trading problem, as indicated in [17] and [18]. This is because the trading environment is too complex to be approximated in a discrete space. On the other hand, in typical Q-learning, the definition of value function always involves a term recoding the future discounted returns [17]. The nature of trading requires to count the profits in an online manner. Not any kind of future market information is allowed in either the sensory part or policy making part of a trading system. While value-function-based methods are plausible for the offline scheduler problems [15], they are not ideal for dynamic online trading [17], [19]. Accordingly, rather than learning the value functions, a pioneering work [17] suggests learning the actions directly that falls into the actor-based framework.

The actor-based RL defines a spectrum of continuous actions directly from a parameterized family of policies. In typical value-function-based method, the optimization always relies on some complicated dynamic programming to derive optimal actions on each state. The optimization of actor-based learning is much simpler that only requires a differentiable objective function with latent parameters. In addition, rather than describing diverse market conditions with some discrete states (in Q-learning), the actor-based method learns the policy directly from the continuous sensory data (market features). In conclusion, the actor-based method exhibits two advantages: 1) flexible objective for optimization and 2) continuous descriptions of market condition. Therefore, it is a better framework for trading than the Q-learning approaches. In [17] and [19], the actor-based learning is termed DRL and we will also use DRL here for consistency.

While the DRL defines a good trading model, it does not shed light on the side of feature learning. It is known that robust feature representation is vital to machine learning performances. In the context of the stock data learning, various feature representation strategies have been proposed from multiple views [20]–[22]. Failure in the extraction of robust features may adversely affect the performances of a trading system on handling market data with high uncertainties. In the field of direct reinforcement trading (DRT), Deng *et al.* [19] attempt to introduce the sparse coding model as a feature extractor for financial analysis. The sparse features achieve much more reliable performances than the DRL to trade stock-IFs.

While admitting the general effectiveness of sparse coding for feature learning [23]–[25], [36], it is essentially a shallow data representation strategy whose performance is not comparable with the state-of-the-art DL in a wide range tests [26], [27]. DL is an emerging technique [28] that allows robust feature learning from big data. The successes of DL techniques have been witnessed in image categorization [26] and speech recognition [29]. In these applications, DL mainly serves to automatically discover informative features from a large amount of training samples. However, to the best of our knowledge, there is hardly any existing work about DL for financial signal mining. This paper will try to generalize the power of DL into a new field for financial signal processing and learning. The DL model will be combined with DRL to design a real-time trading system for financial assert trading.

III. Direct Deep Reinforcement Learning

A. Direct Reinforcement Trading

We generally review Moody's DRL framework [30] here, and it will become clear that typical DRL is essentially a one-layer RNN. 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 $z_t = p_t - p_{t-1}$. Based on the current market conditions, the real-time trading decision (policy) $\delta_t \in \{\text{long, neutral, short}\} = \{1, 0, -1\}$ is made on each time point t. With the symbols defined above, the profit R_t made by the trading model is obtained by

$$R_t = \delta_{t-1} z_t - c|\delta_t - \delta_{t-1}|. \tag{1}$$

In (1), the first term is the profit/loss made from the market fluctuations and the second term is the TC when flipping trading positions at time point t. This TC (c) is the mandatory fee paid to the brokerage company only if $\delta_t \neq \delta_{t-1}$. When two consecutive trading decisions are the same, i.e., $\delta_t = \delta_{t-1}$, no TC is applied there.

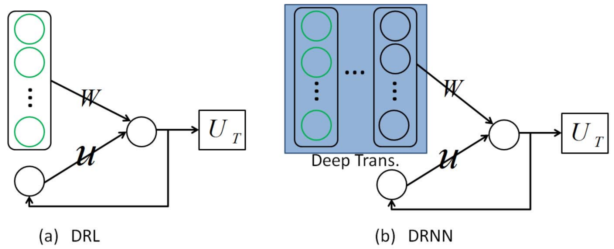
The function in (1) is the value function defined in the typical DRL frameworks. When getting the value function in each time point, the accumulated value throughout the whole training period can be defined as

$$\max_{\Theta} U_T\{R_1 \ldots R_T | \Theta\} \tag{2}$$

where $U_T\{\cdot\}$ is the accumulated rewards in the period of $1, \ldots, T$. Intuitively, the most straightforward reward is the TP made in the T period, i.e., $U_T = \sum_{t=1}^{T} R_t$. Other complicated reward functions, e.g., the risk adjusted returns, can also be used here as the RL objective. For the ease of model explanations, we prefer to use the TP as the objective function in the next parts. Others will be discussed in Section V.

With the well-defined reward function, the primary problem is how to solve it efficiently. In the conventional RL works, the value functions defined in the discrete space are directly iterated by dynamic programming. However, as indicated in [17] and [19], learning the value function directly is not plausible for the dynamic trading problem, because complicated market conditions are hard to be explained within some discrete states. Accordingly, a major contribution of [17] is to introduce a reasonable strategy to learn the trading policy directly. This framework is termed DRL. In detail, a nonlinear function is adopted in DRL to approximate the trading action (policy) at each time point by

$$\delta_t = \tanh[\langle \mathbf{w}, \mathbf{f}_t \rangle + b + u\delta_{t-1}]. \tag{3}$$



(a) DRL

(b) DRNN

Fig. 1. Comparisons of DRL and the proposed DRNN for joint feature learning and DRT.

In the bracket of (3), $\langle \cdot, \cdot \rangle$ is the inner product, $\mathbf{f}_t$ defines the feature vector of the current market condition at time t, and $(\mathbf{w}, b)$ are the coefficients for the feature regression. In DRL, the recent m return values are directly adopted as the feature vector

$$\mathbf{f}_t = [z_{t-m+1}, \ldots, z_t] \in \mathbb{R}^m. \tag{4}$$

In addition to the features, another term $u\delta_{t-1}$ is also added into the regression to take the latest trading decision into consideration. This term is used to discourage the agent to frequently change the trading positions and, hence, to avoid heavy TCs. With the linear transformation in the brackets, $\tanh(\cdot)$ further maps the function into the range of $(-1, 1)$ to approximate the final trading decision. The optimization of DRL aims to learn such a family of parameter set $\Theta = \{\mathbf{w}, u, b\}$ that can maximize the global reward function in (2).

B. Deep Recurrent Neural Network for DDR

While we have introduced the DRL in a regression manner, it is interesting to note that it is in fact an one-layer neural network, as shown in Fig. 1(a). The bias term is not explicitly drawn in the diagram for simplicity. In practical implementations, the bias term can be merged into the weight $\mathbf{w}$ by expanding one dimension of 1 at the end of the feature vector. The feature vector $\mathbf{f}_t$ (green nodes) is the direct input of the system. The DRL neural network exhibits the recurrent structure that has a link from the output (δ_t) to the input layer. One promising property of the RNN is to incorporate the long time memory into the learning system. DRL keeps the past trading actions in the memory to discourage changing trading positions frequently. The system in Fig. 1(a) exploits an RNN to recursively generate trading decisions (learning policy directly) by exploring an unknown environment. However, an obvious pitfall of the DRL is the lack of a feature learning part to robustly summarize the noisy market conditions.

To implement feature learning, in this paper, we introduce the prevalent DL into DRL for simultaneously feature learning and dynamic trading. DL is a very powerful feature learning framework whose potentials have been extensively demonstrated in a number of machine learning problems. In detail, DL constructs a DNN to hierarchically transform the information from layer to layer. Such deep representation encourages much informative feature representations for a specific learning task. The deep transformation has also been found in the neuroscience society when investigating knowledge discovery mechanisms in the brain [31], [32].

These findings further establish the biological theory to support the wide successes of DL.

By extending DL into DRL, the feature learning part (blue panel) is added to the RNN in Fig. 1(a) forming a deep recurrent neural network (DRNN) in Fig. 1(b). We define the deep representation as $\mathbf{F}_t = g_d(\mathbf{f}_t)$, which is obtained by hierarchically transforming the input vector $\mathbf{f}_t$ through the DNN with a nonlinear mapping $g_d(\cdot)$. Then, the trading action in (3) is now subject to the following equation:

$$\delta_t = \tanh[\langle \mathbf{w}, g_d(\mathbf{f}_t)\rangle + b + u\delta_{t-1}]. \tag{5}$$

In our implementation, the deep transformation part is configured with multiple well-connected hidden layers implying that each node on the $(l+1)$th layer is connected to all the nodes in the lth layer. For ease of explanation, we define a_i^l as the input of the ith node on the lth layer and o_i^l is its corresponding output

$$a_i^l = \langle \mathbf{w}_i^l, \mathbf{o}^{(l-1)}\rangle + b_i^l, \quad o_i^l = \frac{1}{1+e^{-a_i^l}} \tag{6}$$

where $\mathbf{o}^{(l-1)}$ are the outputs of all the nodes on the $(l-1)$th layer. The parameter $\langle \mathbf{w}_i^l, b_i^l\rangle$, $\forall i$ are the layerwise latent variables to be learned in the DRNN. In our setting, we set the number of hidden layers in the deep transformation part to 4 and the node number per hidden layer is fixed to 128.

C. Fuzzy Extensions to Reduce Uncertainties

The deep configuration well addresses the feature learning task in the RNN. However, another important issue, i.e., data uncertainty in financial data, should also be carefully considered. Unlike other types of signals, such as images or speech, financial sequences contain high amount of unpredictable uncertainty due to the random gambling behind trading. Besides, a number of other factors, e.g., global economic atmosphere and some company rumors, may also affect the direction of the financial signal in real time. Therefore, reducing the uncertainties in the raw data is an important approach to increase the robustness for financial signal mining.

In the artificial intelligence community, fuzzy learning is an ideal paradigm to reduce the uncertainty in the original data [33], [34]. Rather than adopting precise descriptions of some phenomena, fuzzy systems prefer to assign fuzzy linguist values to the input data. Such fuzzified representations can be easily obtained by comparing the real-world data with a number of fuzzy rough sets and then deriving the corresponding fuzzy membership degrees. Consequently, the learning system only works with these fuzzy representations to make robust control decisions.

For the financial problem discussed here, the fuzzy rough sets can be naturally defined according to the basic movements of the stock price. In detail, the fuzzy sets are defined on the increasing, decreasing, and the no trend groups. The parameters in the fuzzy membership function can then be predefined according to the context of the discussed problem. Alternatively, they could be learned in a fully data-driven manner. The financial problem is highly complicated and it is hard to manually set up the fuzzy membership functions according to the experiences. Therefore, we prefer to directly learn the membership functions and this idea will be detailed in Section IV.

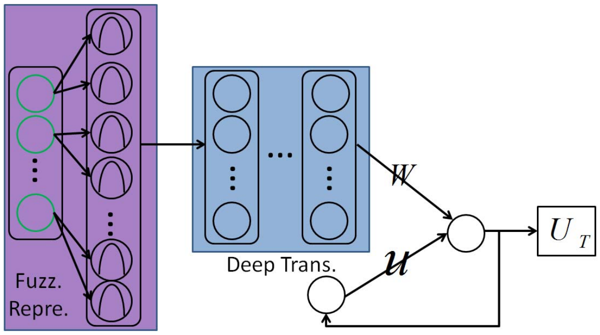


Fig. 2. Overview of fuzzy DRNNs for robust feature learning and self-taught trading.

In fuzzy neural networks, the fuzzy representation part is conventionally connected to the input vector $\mathbf{f}_t$ (green nodes) with different membership functions [35]. To note, in our setting, we follow a pioneering work [35] to assign k different fuzzy degrees to each dimension of the input vector. In the cartoon of Fig. 2, only two fuzzy nodes ($k = 2$) are connected to each input variable due to the space limitation. In our practical implementation, k is fixed as 3 to describe the increasing, decreasing, and no trend conditions. Mathematically, the ith fuzzy membership function $\upsilon_i(\cdot) : R \rightarrow [0, 1]$ maps the ith input as a fuzzy degree

$$o_i^{(l)} = \upsilon_i\left(a_i^{(l)}\right) = e^{-\left(a_i^{(l)}-m_i\right)^2/\sigma_i^2} \quad \forall i. \tag{7}$$

The Gaussian membership function with mean m and variance σ^2 is utilized in our system following the suggestions of [37] and [38]. After getting the fuzzy representations, they are directly connected to the deep transformation layer to seek for the deep transformations.

In conclusion, the fuzzy DRNN (FDRNN) is composed of three major parts as fuzzy representation, deep transformation, and DRT. When viewing the FDRNN as a unified system, these three parts, respectively, play the roles of data preprocessing (reduce uncertainty), feature learning (deep transformation), and trading policy making (RL). The whole optimization framework is given as follows:

$$\begin{aligned}
\max_{\{\Theta, g_d(\cdot), \upsilon(\cdot)\}} \quad & U_T(R_1..R_T) \\
\text{s.t.} \quad & R_t = \delta_{t-1}z_t - c|\delta_t - \delta_{t-1}| \\
& \delta_t = \tanh(\langle \mathbf{w}, \mathbf{F}_t\rangle + b + u\delta_{t-1}) \\
& \mathbf{F}_t = g_d(\upsilon(\mathbf{f}_t))
\end{aligned} \tag{8}$$

where there are three groups of parameters to be learned, i.e., the trading parameters $\Theta = (\mathbf{w}, b, u)$, fuzzy representations $\upsilon(\cdot)$, and deep transformations $g_d(\cdot)$. In the above optimization, U_T is the ultimate reward of the RL function, δ_t is the policy approximated by the FRDNN, and $\mathbf{F}_t$ is the high-level feature representation of the current market condition produced by DL.

IV. DRNN Learning

While the optimization in (8) is conceptually elegant, it unfortunately leads to a relative difficult optimization. This is because the configured complicated DNN involves thousands of latent parameters to be inferred. In this section, we propose a practical learning strategy to train the DNN using two steps of system initialization and fine tuning.

A. System Initializations

Parameter initialization is a critical step to train a DNN. We will introduce the initialization strategies for the three learning parts. The fuzzy representation part [Fig. 2 (purple panel)] is easily initialized. The only parameters to be specified are the fuzzy centers (m_i) and widths (σ_i^2) of the fuzzy nodes, where i means the ith node of the fuzzy membership layer. We directly apply k-means to divide the training samples into k classes. The parameter k is fixed as 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 σ_i^2.

The AE is adopted to initialize the deep transformation part in Fig. 2 (blue panel). In a nutshell, AE aims at optimally reconstructing the input information on a virtual layer placed after the hidden representations. For ease of explanation, three layers are specified here, i.e., the (l)th input layer, the $(l+1)$th hidden layer, and the $(l+2)$th reconstruction layer. These three layers are all well connected. We define $h_\theta(\cdot)$ [respectively, $h_\gamma(\cdot)$] as the feedforward transformation from the lth to $(l+1)$th layer [respectively, $(l+1)$th to $(l+2)$th layer] with parameter set θ (respectively, γ). The AE optimization minimizes the following loss:

$$\sum_t \|\mathbf{x}_t^{(l)} - h_\gamma\left(h_\theta\left(\mathbf{x}_t^{(l)}\right)\right)\|_2^2 + \eta\|\mathbf{w}^{(l+1)}\|_2^2. \tag{9}$$

To note, $\mathbf{x}_t^{(l)}$ are the nodes' statuses of the lth layer with the tth training sample as input. In (9), a quadratic term is added to avoid the overfitting phenomena. After solving the AE optimization, parameter set $\theta = \{\mathbf{w}^{(l+1)}, b^{(l+1)}\}$ is recorded in the network as the initialized parameter of the $(l+1)$th layer. The reconstruction layer and its corresponding parameters γ are not used. This is because the reconstruction layer is just a virtual layer, assisting parameter learning of the hidden layer [28], [39]. The AE optimizations are implemented on each hidden layer sequentially until all the parameters in the deep transformation part have been set up.

In the DRL part, the parameters can be initialized using final deep representation $\mathbf{F}_t$ as the input to the DRL model. This process is equivalent to solving the shallow RNN in Fig. 1(a), which has been discussed in [17]. It is noted that all the learning strategies presented in this section are all about parameter initializations. In order to make the whole DL system perform robustly in addressing difficult tasks, a fine tuning step is required to precisely adjust the parameters of each layer. This fine tuning step can be considered as task-dependent feature learning.

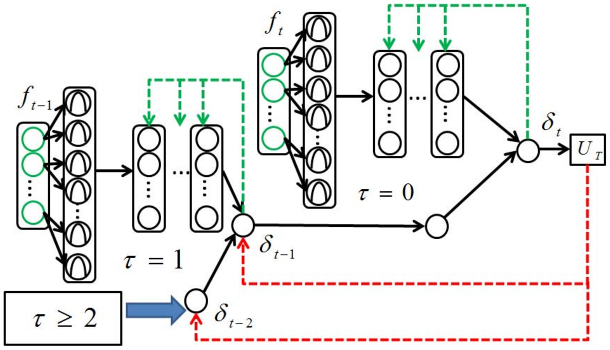


Fig. 3. Task-aware BPTT for RDNN fine tuning.

B. Task-Aware BPTT

In the conventional way, the error BP method is applied to the DNN fine tuning step. However, the FRDNN is bit complicated that exhibits both recurrent and deep structures. We denote θ as the general parameter in the FRDNN, and its gradient is easily calculated by the chain rule

$$\frac{\partial U_T}{\partial \theta} = \sum_t \frac{dU_t}{dR_t}\left\{\frac{dR_t}{d\delta_t}\frac{d\delta_t}{d\theta} + \frac{dR_t}{d\delta_{t-1}}\frac{d\delta_{t-1}}{d\theta}\right\}$$
$$\frac{d\delta_t}{d\theta} = \frac{\partial \delta_t}{\partial \theta} + \frac{\partial \delta_t}{\partial \delta_{t-1}}\frac{d\delta_{t-1}}{d\theta}. \tag{10}$$

From (10), it is apparent when deriving the gradient $d\delta_t/d\theta$, one should recursively calculate the gradient for $d\delta_{t-\tau}/d\theta$, $\forall\tau = 1, \ldots, T$.[1] Such recursive calculation inevitably imposes great difficulties for gradient derivations. To simplify the problem, we introduce the famous BPTT [40] method to cope with the recurrent structure of the NN.

By analyzing the FRDNN structure in Fig. 2, the recurrent link comes from the output side to the input side, i.e., δ_{t-1} is used as the input of the neuron to calculate δ_t. Fig. 3 shows the first two-step unfolding of the FRDNN. We call each block with different values of τ as a time stack, and Fig. 3 shows two time stacks (with $\tau = 0$ and $\tau = 1$). After the BPTT unfolds, the current system does not involve any recurrent structure and the typical BP method is easily applied to it. When getting parameters' gradients at each separate time stack, they are averaged together forming the final gradient of each parameter.

According to Fig. 3, the original DNN becomes even deeper due to the implementations of time-based unfolding. To clarify this point, we remind the readers to notice the time stacks after expansion. It leads to a deep structure along different time delays. Moreover, every time stack (with different values of τ) contains its own deep feature learning part. When directly applying the BPTT, the gradient vanish on deep layers is not avoided in the fine-tuning step [41]. This problem becomes even worse on the high-order time stacks and the front layers.

To solve the aforementioned problem, we propose a more practical solution to bring the gradient information directly from the learning task to each time stack and each layer of the DL part. In the time unfolding part, the red dotted lines

[1] In (10), while the calculation only explicitly relies on $d\delta_{t-1}/d\theta$, the term $d\delta_{t-1}/d\theta$ further depends on $d\delta_{t-2}/d\theta$ making the calculations recursively evolved.

Algorithm 1 Training Process for the FRDNN

```
Input          : Raw price ticks $p_1, \ldots, p_T$ received in an
                 online manner; $\rho, c_0$ (learning rate).
Initialization : Initialize the parameters for the fuzzy
                 layers (by fuzzy clustering), deep layers
                 (auto-encoder) and reinforcement learning
                 part sequentially.
1  repeat
2      c = c + 1;
3      Update learning rate $\rho_c = \min(\rho, \rho\frac{c_0}{c})$ for this outer
       iteration;
4      for $t = 1 \ldots T$ do
5          Generate Raw feature $\mathbf{f}_t$ vector from price ticks;
6          BPTT: Unfold the RNN at time $t$ into $\tau + 1$ stacks;
7          Task-aware Propagation: Add the virtual links from
           the output to each deep layer;
8          BP: Back-propagate the gradient through the
           unfolded network as in Fig. 3;
9          Calculated $\nabla(U_t)_\Theta$ by averaging its gradient values
           on all the time stacks.;
10         Parameter Updating: $\Theta_t = \Theta_{t-1} - \rho_c \frac{\nabla(U_t)_\theta}{||\nabla(U_t)_\theta||}$;
11     end
12 until convergence;
```

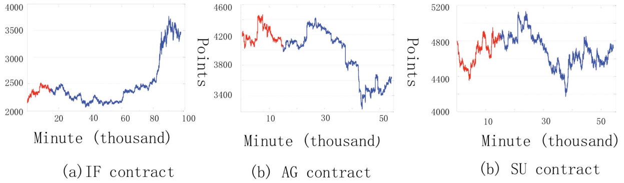
are connected from the task U_T to the output node of each time stack. With this setting, the back-propagated gradient information of each time stack comes from two respective parts: 1) the previous time stack (lower order time delay) and 2) the reward function (learning task). Similarly, the gradient of the output node in each time stack is brought back to the DL layers by the green dotted line. Such a BPTT method with virtual lines connecting with the objective function is termed task-aware BPTT.

The detailed process to train the FRDNN has been summarized in Algorithm 1. In the algorithm, we denote Θ as the general symbol to represent parameters. It represents the whole latent parameters' family involved in the FRDNN. Before the gradient decreasing implementation in line 10, the calculated gradient vector is further normalized to avoid extremely large value in the gradient vector.

V. Experimental Verifications

A. Experimental Setup

We test the DDR trading model on the real-world financial data. Both the stock index and commodity future contracts are tested in this section. For the stock-index data, we select the stock-IF contract, which is the first index-based future contract traded in China. The IF data is calculated based on the prices of the top 300 stocks from both Shanghai and Shenzhen exchange centers. The IF future is the most liquid one and occupies the heaviest trading volumes among all the future contracts in China. On the commodity market, the silver (AG) and sugar (SU) contracts are used, because both of them exhibit very high liquidity, allowing trading actions to be executed in almost real time. All these contracts allow both short and long operations. The long (respectively, short) position makes profits when the subsequent market price goes higher (respectively, lower).



(a) IF contract (b) AG contract (b) SU contract

Fig. 4. Prices (based on minute resolutions) of the three tested future contracts. Red parts: RDNN initializations. Blue parts: out-of-sample tests.

TABLE I

Summary of Some Practical Properties of the Traded Contracts

Contract	Periods	CNY/pnt	TC	c
IF	1/14–9/15	300	30 (0.1pnt)	1pnt
AG	1/14–1/15	15	5 (0.4pnt)	2pnt
SU	1/14–1/15	5	3 (0.3pnt)	1.5pnt

The financial data are captured by our own trading system in each trading day and the historic data is maintained in a database. In our experiment, the minute-level close prices are used, implying that there is a 1-min interval between the price p_t and p_{t+1}. The historic data of the three contracts in the minute resolutions are shown in Fig. 4. In this one-year period, the IF contracts accumulate more ticks than commodity data because the daily trading period of IF is much longer than commodity contracts.

From Fig. 4, it is also interesting to note that these three contracts exhibit quite different market patterns. IF data get very large upward and downward movements in the tested period. The AG contract, generally, shows a downward trend and the SU has no obvious direction in the testing period. For practical usage, some other issues related to trading should also be considered. We have summarized some detailed information about these contracts in Table I. The inherent values of these three contracts are evaluated by China Yuan (CNY) per point (CNY/pnt). For instance, in the IF data, the increase (decrease) in one point may lead to a reward of 300 CNY for a long (respectively, short) position and vice versa. The TCs charged by the brokerage company is also provided. By considering other risky factors, a much higher c is set in (1). It is five times higher than the real TCs.

The raw price changes of the last 45 min and the momentum change to the previous 3 h, 5 h, 1 day, 3 days, and 10 days is directly used as the input of the trading system ($\mathbf{f}_t \in \mathbb{R}^{50}$). In the fuzzy learning part, each of the 50 input nodes are connected with three fuzzy membership functions to seek for the first-level fuzzy representation in $\mathbb{R}^{150}$. Then, the fuzzy layer is sequentially passed through four deep transformation layers with 128, 128, 128, and 20 hidden nodes per layer. The feature representation ($\mathbf{F}_t \in \mathbb{R}^{20}$) of the final deep layer is connected with the DRL part for trading policy making.

B. Details on Deep Training

In this section, we discuss some details related to deep training. In practice, the system is trained by two sequential steps of initialization and online updating. In the initialization step, the first 15 000 time points of each time series in Fig. 4 (red parts) are employed for system warming up. It is noted that these initialization data will not be used for out-of-sample tests. After initialization, the parameters in the RDNN are iteratively updated in an online manner with the recently released data. The online updating strategy allows the model to get aware of the latest market condition and revise its parameters accordingly.

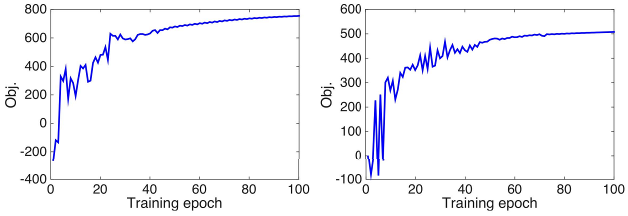
In practice, the first 15 000 time points are used to set up the RDNN and the well-trained system is exploited to trade the time points from 15 001 to 20 000. Then, the sliding window of the training data is moved 5000 ticks forward covering a new training set from 5000 to 20 000. As indicated in Section IV, the training phase of the RDNN is composed of two main steps of layerwise parameter initialization and fine tuning. It is clarified here that the parameter initialization implementations are only performed in the first round of training, i.e., on the first 15 000 ticks. With the sliding window of the training set moving ahead, the optimal parameters obtained from the last training round are directly used as the initialized values.

FDDR is a highly nonconvex system and only a local minimum is expected after convergence. Besides, the overfitting phenomenon is a known drawback faced by most DNNs. To mitigate the disturbances of overfitting, we adopt two convenient strategies that have been proved to be powerful in practice. The first strategy is the widely used early stopping method for DNN training. The system is only trained for 100 epochs with a gradient decreasing parameter be $\eta_c = 0.97$ in Algorithm 1. Second, we conduct model selection to select a good model for the out-of-sample data. To achieve this goal, 15 000 training points are divided into two sets as RDNN training set (first 12 000) and validation set (last 3000). On the first 12 000 time points, FDDR is trained for 5 times and the best one is selected on the next 3000 blind points. Such validation helps to exclude some highly overfitted NNs from the training set.

Another challenge in training RDNN comes from the gradient vanishing issue, and we have introduced the task-aware BPTT method to cope with this problem. To prove its effectiveness, we show the training performance with the comparison with the typical BPTT method. The trading model is trained on the first 12 000 points of the IF data in Fig. 4(a). The objective function values (accumulated rewards) of the two methods along with the training epochs are shown in Fig. 5. From the comparisons, it is apparent that the task-aware BPTT outperforms the BPTT method in making more rewards (accumulated trading profits). Moreover, the task-aware BPTT requires less iterative steps for convergence.

C. General Evaluations

In this section, we evaluate the DDR trading system on practical data. The system is compared with other RL systems for online trading. The first competitor is the DRL system [17]. Besides, the sparse coding-inspired optimal training (SCOT) system [19] is also evaluated, which uses shallow feature learning part of sparse coding.[2] Finally, the results of DDR and its FDDR are also reported.



(a) Task-aware BPTT (b) BPTT

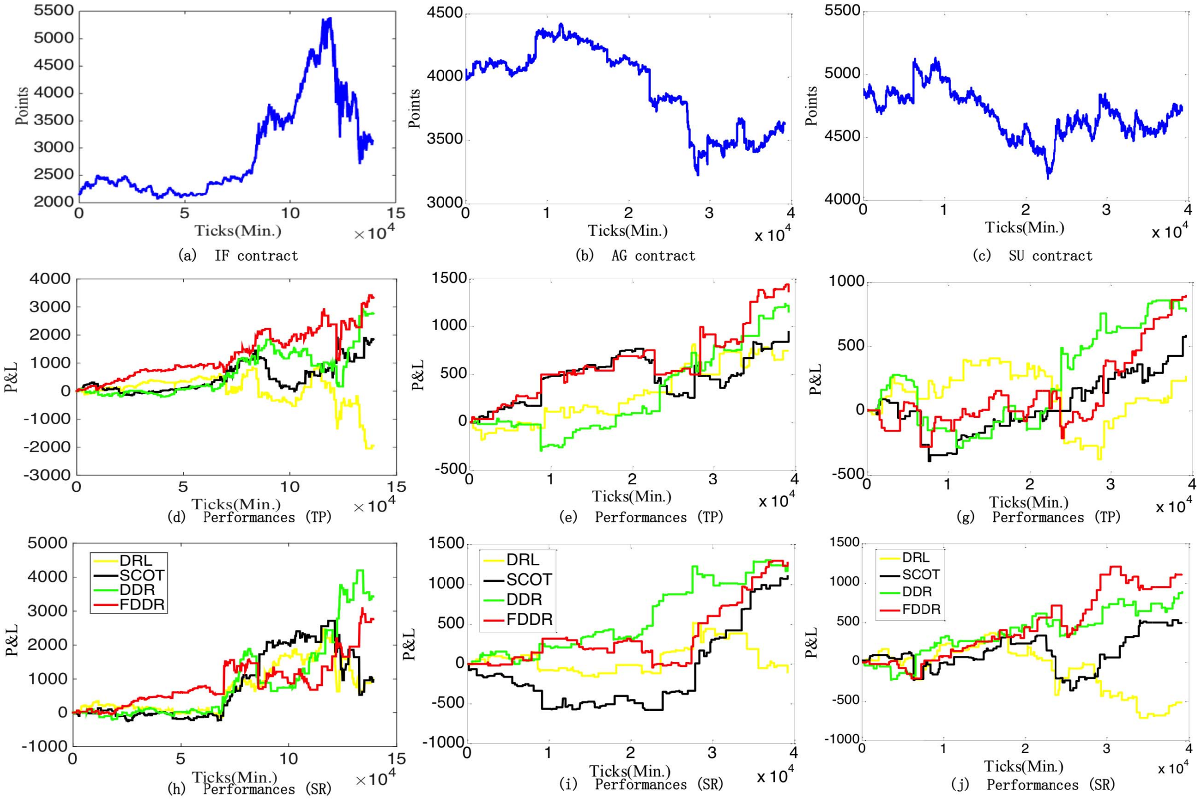
Fig. 5. Training epochs and the corresponding rewards by training the DRNN by (a) task-aware BPTT and (b) normal BPTT. The training data are the first 12 000 data points in Fig. 4(a).

In the previous discussions, the reward function defined for RL is regarded as the TPs gained in the training period. Compared with total TP, in modern portfolio theory, the risk-adjusted profits are more widely used to evaluate a trading system's performance. In this paper, we will also consider an alternative reward function into the RL part, i.e., SR, which has been widely used in many trading related works [42], [43]. The SR is defined as the ratio of average return to standard deviation of the returns calculated in period $1, \ldots, T$, i.e., $U_T^{\mathrm{SR}} = (\mathrm{mean}(R_t)/\mathrm{std}(R_t))$. To simplify the expression, we follow the same idea in DRL [17] to use the moving SR instead. In general, moving SR gets the first-order Taylor expansion of typical SR and then updates the value in an incremental manner. Please refer to [17, Sec. 2.4] for detailed derivations. Different trading systems are trained with both TP and SR as the RL objectives. The details of the profit and loss (P&L) curves are shown in Fig. 6. The quantitative evaluations are summarized in Table II, where the testing performances are also reported in the forms of TP and SR. The performance of buying and holding (B&H) is also reported in Table II as the comparison baseline. From the experimental results, three observations can be found as follows.

The first observation is that all the methods achieved much more profits in the trending market. Since we have allowed short operation in trading, the trader can also make money in the downward market. This can be denmonstrated from the IF data that Chinese market exhibits an increasing trend in the first, followed by a sudden drop. FDDR makes profits in either case. It is also observed that the P&L curve suffers a drawback during the transition period from the increasing trend to the decreasing trend. This is possible due to the significant differences in the training and testing data. In general, the RL model is particularly suitable to be applied to the trending market condition.

Second, FDDR and DDR generally outperform the other two competitors on all the markets. SCOT also gains better performance than DRL in most conditions. This observation verifies that feature learning indeed contributes in improving the trading performance. Besides, the DL methods

[2] The basis in the sparse coding dictionary is set to 128, which is the same as the hidden node number of the RDNN.



(a) IF contract (b) AG contract (c) SU contract
(d) Performances (TP) (e) Performances (TP) (g) Performances (TP)
(h) Performances (SR) (i) Performances (SR) (j) Performances (SR)

Fig. 6. Testing future data (top) and the P&L curves of different trading systems with TPs (middle) and SR (bottom) as RL objectives, respectively.

TABLE II
PERFORMANCES OF DIFFERENT RL SYSTEMS ON DIFFERENT MARKETS

	IF				AG				SU			
	TP		SR		TP		SR		TP		SR	
	TP	SR(%)	TP	SR(%)	TP	SR(%)	TP	SR(%)	TP	SR(%)	TP	SR(%)
BH	739	-	-	-	-415	-	-	-	-121	-	-	-
DRL	-1998.2	-5.4	1054.4	3.1	-113	-2.4	745	12.2	271	4.3	-514	-7.4
SCOT	1873.2	5.6	1056.2	3.5	559	6.5	953	16.2	590	11.2	492	7.8
DDR	2785.4	9.3	3424.4	12.9	985	16.4	1147	19.4	787	13.0	898	15.8
FDDR	3256.6	11.2	2760.2	10.0	1567	21.2	1335	20.0	901	13.6	1118	19.3

(FDDR and DDR) make more profits with higher SR on all the tests than the shallow learning approach (SCOT). Among the two DL methods, adding an extra layer for fuzzy representation seems to be a good way to further improve the results. This claim can be easily verified from Table II and Fig. 6 in which FDDR wins the DDR on all the tests except the one in Fig. 6(g). As claimed in the last paragraph, the DRL is a trend following system that may suffer losses on the market with small volatility. However, it is observed from the results that even on the nontrending period, e.g., on the SU data or in the early period of the IF data, FDDR is also effective to make the positive accumulation from the swing market patterns. Such finding successfully verifies another important property of fuzzy learning in reducing market uncertainties.

Third, exploiting SR as the RL objective always leads to more reliable performances. Such reliability can be observed from both the SR quantity in Table II and the shapes of the P&L curves in Fig. 6. It is observed from Table II that the highest profits on the IF data were made by optimizing the TP as the objective in DDR. However, the SR on that testing condition is worse than the other. In portfolio management, rather than struggling for the highest profits with high risk, it is more intellectual to make good profits within acceptable risk level. Therefore, in practical usage, it is still recommended to use the SR as the reward function for RL.

In conclusion, the RL framework is perhaps a trend-based trading strategy and could make reliable profits on the markets with large price movement (no matter in which direction). The DL-based trading systems generally outperform other DRL models either with or without shallow feature learning. By incorporating the fuzzy learning concept into the system, the FDDR can even generate good results in the nontrending period. When training a deep trading model, it is suggested to use the SR as the RL objective which balances the profit and the risk well.

TABLE III
COMPARISONS WITH OTHER PREDICTION-BASED DNNS

	PR	TT	Profits with different costs		
			0	1	2
CDNN	0.523	2863	4362.6	1499.6	-1499.4
RNN	0.529	3232	5162.4	1930.4	-1301.6
LSTM	0.535	3145	5427.2	2282.2	-822.8
FDDR	0.512	376	3652.4	2760.2	774.2

D. Comparisons With Prediction-Based DNNs

We further compare the DDR framework with other prediction-based NNs [1]. The goal of the prediction-based NN is to predict whether the closed price of the next bar (minute resolution) is going higher, lower, or suffering no change. The three major competitors are convolutional DNN (CDNN) [26], RNN [1], and long short-term memory (LSTM) [44] RNNs. To make fair comparisons, the same training and testing strategies in FDDR are applied to them.

We sought to the python-based DL package *Keras*[3] to implement the three comparison methods. *Keras* provides the benchmark implementations of convolution, recurrent, and LSTM layers for public usages. The CDNN is composed of five layers: 1) an input layer (50 dimension input); 2) a convolutional layer with 64 convolutional kernels (each kernel is of length 12); 3) a max pooling layer; 4) a fully connected dense layer; and 5) a soft-max layer with three outputs. The RNN contains an input layer, a dense layer (128 hidden neurons), a recurrent layer, and a soft-max layer for classification. The LSTM-RNN shares the same configuration as RNN except for replacing the recurrent layer with the LSTM module.

In practical implementation, for the prediction-based learning systems, only the trading signal with high confidence was trusted. This kind of signal was identified if the predicted probability for one direction (output of the soft-max function) was higher than 0.6. We have reported profitable rate (PR), trading times (TTs), and the TPs with different trading costs in Table III. The PR is calculated by dividing the number of profitable trades with the total trading number (TT). The results were obtained on the IF market.

It is observed from the table that all the learning systems' PRs are only slightly better than 50%. Such low PR implies the difficulty of accurately predicting the price movement in the highly dynamic and complicated financial market. However, the low PR does not equivalently mean no trading opportunity. For instance, consider a scenario where a wining trade could make two points while a losing trade averagely lost one point. In such a market, even a 40% PR could substantially lead to positive net profits. Such a phenomenon has been observed from our experiment that all the methods accumulate quite reliable profits from the IF data when there is zero trading cost. The recurrent machines (RDNN and LSTM) make much higher prediction accuracy than others under this condition.

However, when taking the practical trading costs into considerations, the pitfalls of prediction-based DNN become apparent. By examining the total TTs (TT column) in Table III, the prediction systems potentially change trading positions quite more often than our FDDR. When the trading costs increase to two points, only the FDDR could make positive profits while other systems suffer a big loss due to the heavy TCs. This is because prediction-based systems only consider the market condition to make decisions. In FDDR, we have considered both current market condition and the trading actions to avoid heavy trading costs. This is the benefit of learning the market condition and the trading decision in a joint framework.

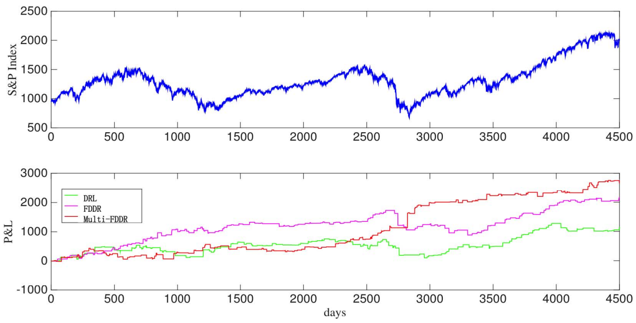


Fig. 7. Testing S&P data and the P&L curves of different trading systems.

It is observed from the data that the training data and the testing data may not share the same pattern. Financial signal does not like other stationary or structured sequential signals, such as music, that exhibit periodic and repeated patterns. Therefore, in this case, we abandoned the conventional RNN configurations that recursively remembers the historical feature information. Instead, the proposed FDDR only takes the current market condition and the past trading history into consideration. Memorizing the trading behavior helps the system to maintain a relative low position-changing frequency to avoid heavy TCs [17].

E. Verifications on the Global Market

The performances of FDDR were also verified on the global market to trade the S&P 500 index. The day-resolution historic data of S&P from January 1990 to September 2015 were obtained from Yahoo Finance covering more than 6500 days in all. In this market, we directly used the previous 20 days' price changes as the raw feature. The latest 2000 trading days (about eight years) were used to train FDDR and the parameters were updated every 100 trading days. In this case, we fix the trading cost to 0.1% of the index value. The performances of different RL methods on the S&P daily data from November 1997 to September 2015 are provided in Fig. 7. From the results, it is observed that the proposed FDDR could also work on the daily S&P index. When compared with the green curve (DRL), the purple curve (FDDR) makes much more profits. The DL framework makes about 1000 points more than the shallow DRL.

It is also interesting to note that the U.S. stock market is heavily influenced by the global economics. Therefore, we also considered an alternative way to generate features for FDDR learning. The index changes of other major countries

[3] http://www.keras.io

TABLE IV
ROBUSTNESS VERIFICATIONS WITH DIFFERENT DNN SETTINGS

	Layers number	$l = 3$		$l = 4$		$l = 5$	
Time stacks	Nodes number	TP	Training cost	TP	Training cost	TP	Training cost
$\tau = 2$	$N = 64$	546	3.1	986	7.2	1027	12.4
	$N = 128$	633	19.2	1395	24.2	1466	39.8
	$N = 256$	662	27.7	1378	35.6	1673	61.3
$\tau = 4$	$N = 64$	478	5.8	1021	11.5	1123	19.3
	$N = 128$	672	27.1	1456	45.5	1386	54.2
	$N = 256$	694	39.6	1293	51.7	1533	72.4

were also provided to FDDR. The relative indices include the FTSE100 index (U.K.), Hangseng Index (Hong Kong), Nikkei 225 (Japan), and Shanghai Stock Index (China). Since the trading days of different countries may not be exactly the same, we adopted the trading days of S&P as the reference to align the data. On other markets, the missing trading days' price changes were filled with zero. The latest 20 days' price changes of the five markets were stacked as a long input vector ($\mathbb{R}^{100}$) for FDDR.

The P&L of the FDDR using multimarket features (multi-FDDR) was provided as the red curve in Fig. 7. It is interesting to note that the FDDR generally outperforms Multi-FDDR before the 2800th point (January 2010). However, after that, the Multi-FDDR performs much better. It may be due to the fact that more algorithmic trading companies have participated into the market after the year of 2010. Therefore, the raw price changes may not be that informative as before. In such a case, simultaneously monitoring multiple markets for decision making is perhaps a smart choice.

F. Robustness Verifications

In this section, we verify the robustness of the FDDR system. The major contribution of this paper is to introduce the DL concept into the DRL framework for feature learning. It is worth conducting some detailed discussions on the feature learning parts and further investigating their effects on the final results. The NN structures studied here include different number of BPTT stacks (τ), different hidden layers (l), and different nodes number per layer (N).

This part of the experiment will be discussed on the practical IF data from January 2014 to January 2015. The number of deep layers is varied from $l = 3$ to $l = 5$, and the number of nodes per layer is tested on three levels of $N = 64$, $N = 128$, and $N = 256$, respectively. We also test the order of BPTT expansion with $\tau = 2$ and $\tau = 4$. At each testing level, the out-of-sample performances and the corresponding training complexities are both reported in Table IV. For the testing performance, we only report the TPs, and the training cost is evaluated in minutes. The computations are implemented on an eight-core 3.2-GHZ computational platform with 16-G RAM.

From the results, it is observed that the computational complexity increases when N becomes large. This is because the nodes of neighboring layers are fully connected. The number of unknown parameters will be largely increased once more nodes are used in one layer. However, this seems to help less in improving the performances by adding N from 128 to 256. Rather than increasing N, an alternative approach is to increase the layer number l as shown in the horizontal direction of Table IV. By analyzing the results with different layer numbers, it is concluded that the depth of the DNN is vital to the feature learning part. This means that the depth of the layers contributes a lot in improving the final performance. The TP has also been significantly improved when the depth increases from 3 to 5. Meanwhile, the computational costs are also increased with more layers. Among all the DNN structures, the best performance is achieved using 5 layers with 256 nodes per layer. The corresponding computational costs are also the heaviest in all the tests.

When analyzing different time expansion orders (τ), we have not found strong evidence to claim that higher BPTT order is good for performance. When using $\tau = 4$, the TP with each DNN setting is quite similar by using $\tau = 2$. However, by extending the length of the BPTT order, the computational complexities have been increased. This phenomenon is perhaps due to the fact that the long-term memory is not as important as the short-term memory. Therefore, the BPTT with lower order is preferred in this case.

According to the previous comparisons, we have selected $\tau = 2$, $N = 128$ and $l = 3$ as the default RDNN setting. While further increasing the layer numbers could potentially improve the performance, the training complexity is also increased. Although the trading problem discussed here is not in a high frequency setting, the training efficiency still needs to be explicitly considered. We, therefore, recommend a relatively simple NN structure to guarantee good performances.

VI. CONCLUSION

This paper introduces the contemporary DL into a typical DRL framework for financial signal processing and online trading. The contributions of the system are twofold. First, it is a technical-indicator-free trading system that greatly releases humans to select the features from a large amount of candidates. This advantage is due to the automatic feature learning mechanism of DL. In addition, by considering the nature of the financial signal, we have extended the fuzzy learning into the DL model to reduce the uncertainty in the original time series. The results on both the stock-index and commodity future contracts demonstrate the effectiveness of the learning system in simultaneous market condition summarization and optimal action learning. To the best of our knowledge, this is the first attempt to use the DL with the real-time financial trading.

While the power of the DDR system has been verified in this paper, there are some promising future directions. First, all the methods proposed in this paper only handle one share of the asset. In some large hedge funds, the trading systems are always required to be capable in managing a number of assets simultaneously. In the future, the DL framework will be extended to extract features from multiple asserts and to learn the portfolio management strategies. Second, the financial market is not stationary that may change in real time. The knowledge learned from the past training data may not sufficiently reflect the information of the subsequent testing period. The method to intelligently select the right training period is still an open problem in the field.

REFERENCES

[1] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch, II, "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks," *IEEE Trans. Neural Netw.*, vol. 9, no. 6, pp. 1456–1470, Nov. 1998.

[2] D. Prokhorov, G. Puskorius, and L. Feldkamp, "Dynamical neural networks for control," in *A Field Guide to Dynamical Recurrent Networks*. New York, NY, USA: IEEE Press, 2001.

[3] D. Zhao and Y. Zhu, "MEC—A near-optimal online reinforcement learning algorithm for continuous deterministic systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 2, pp. 346–356, Feb. 2015.

[4] W. Schultz, P. Dayan, and P. R. Montague, "A neural substrate of prediction and reward," *Science*, vol. 275, no. 5306, pp. 1593–1599, 1997.

[5] H. R. Beom and K. S. Cho, "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, no. 3, pp. 464–477, Mar. 1995.

[6] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng, "Autonomous helicopter flight via reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2003, pp. 799–806.

[8] Y.-D. Song, Q. Song, and W.-C. Cai, "Fault-tolerant adaptive control of high-speed trains under traction/braking failures: A virtual parameter-based approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 2, pp. 737–748, Apr. 2014.

[9] C. J. Neely, D. E. Rapach, J. Tu, and G. Zhou, "Forecasting the equity risk premium: The role of technical indicators," *Manage. Sci.*, vol. 60, no. 7, pp. 1772–1791, 2014.

[10] J. J. Murphy, *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. New York, NY, USA: New York Institute of Finance, 1999.

[11] J. M. Poterba and L. H. Summers, "Mean reversion in stock prices: Evidence and implications," *J. Financial Econ.*, vol. 22, no. 1, pp. 27–59, 1988.

[12] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[13] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998.

[14] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2014.

[15] G. Tesauro, "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural Comput.*, vol. 6, no. 2, pp. 215–219, 1994.

[16] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA, USA: Athena Scientific, 1995.

[17] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Trans. Neural. Netw.*, vol. 12, no. 4, pp. 875–889, Jul. 2001.

[18] M. A. H. Dempster and V. Leemans, "An automated FX trading system using adaptive reinforcement learning," *Expert Syst. Appl.*, vol. 30, no. 3, pp. 543–552, 2006.

[19] Y. Deng, Y. Kong, F. Bao, and Q. Dai, "Sparse coding-inspired optimal trading system for HFT industry," *IEEE Trans. Ind. Informat.*, vol. 11, no. 2, pp. 467–475, Apr. 2015.

[20] K.-I. Kamijo and T. Tanigawa, "Stock price pattern recognition: A recurrent neural network approach," in *Proc. Int. Joint Conf. Neural Netw.*, San Diego, CA, USA, 1990, pp. I-215–I-221.

[21] Y. Deng, Q. Dai, R. Liu, Z. Zhang, and S. Hu, "Low-rank structure learning via nonconvex heuristic recovery," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 3, pp. 383–396, Mar. 2013.

[22] K. K. Ang and C. Quek, "Stock trading using RSPOP: A novel rough set-based neuro-fuzzy approach," *IEEE Trans. Neural. Netw.*, vol. 17, no. 5, pp. 1301–1315, Sep. 2006.

[23] Y. Deng, Y. Liu, Q. Dai, Z. Zhang, and Y. Wang, "Noisy depth maps fusion for multiview stereo via matrix completion," *IEEE J. Sel. Topics Signal Process.*, vol. 6, no. 5, pp. 566–582, Sep. 2012.

[24] Y. Deng, Q. Dai, and Z. Zhang, "Graph Laplace for occluded face completion and recognition," *IEEE Trans. Image Process.*, vol. 20, no. 8, pp. 2329–2338, Aug. 2011.

[25] J. Yang, K. Yu, Y. Gong, and T. Huang, "Linear spatial pyramid matching using sparse coding for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Miami Beach, FL, USA, Jun. 2009, pp. 1794–1801.

[26] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, Montreal, QC, Canada, Jun. 2009, pp. 609–616.

[27] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. Audio, Speech, Language Process.*, vol. 20, no. 1, pp. 30–42, Jan. 2012.

[28] Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

[29] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Vancouver, BC, Canada, May 2013, pp. 6645–6649.

[30] J. Moody, L. Wu, Y. Liao, and M. Saffell, "Performance functions and reinforcement learning for trading systems and portfolios," *J. Forecasting*, vol. 17, nos. 5–6, pp. 441–470, 1998.

[31] J. D. Bransford, A. L. Brown, and R. R. Cocking, *How People Learn: Brain, Mind, Experience, and School*. Washington, DC, USA: National Academy Press, 1999.

[32] T. Ohyama, W. L. Nores, J. F. Medina, F. A. Riusech, and M. D. Mauk, "Learning-induced plasticity in deep cerebellar nucleus," *J. Neurosci.*, vol. 26, no. 49, pp. 12656–12663, 2006.

[33] G. J. Klir and T. A. Folger, *Fuzzy Sets, Uncertainty, and Information*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1988.

[34] N. R. Pal and J. C. Bezdek, "Measuring fuzzy uncertainty," *IEEE Trans. Fuzzy Syst.*, vol. 2, no. 2, pp. 107–118, May 1994.

[35] C.-T. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol. 40, no. 12, pp. 1320–1336, Dec. 1991.

[36] Y. Deng, Y. Li, Y. Qian, X. Ji, and Q. Dai, "Visual words assignment via information-theoretic manifold embedding," *IEEE Trans. Cybern.*, vol. 44, no. 10, pp. 1924–1937, Oct. 2014.

[37] C.-T. Lin, C.-M. Yeh, S.-F. Liang, J.-F. Chung, and N. Kumar, "Support-vector-based fuzzy neural network for pattern classification," *IEEE Trans. Fuzzy Syst.*, vol. 14, no. 1, pp. 31–41, Feb. 2006.

[38] F.-J. Lin, C.-H. Lin, and P.-H. Shen, "Self-constructing fuzzy neural network speed controller for permanent-magnet synchronous motor drive," *IEEE Trans. Fuzzy Syst.*, vol. 9, no. 5, pp. 751–759, Oct. 2001.

[39] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, no. 12, pp. 3371–3408, Dec. 2010.

[40] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[41] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[42] O. Ledoit and M. Wolf, "Robust performance hypothesis testing with the Sharpe ratio," *J. Empirical Finance*, vol. 15, no. 5, pp. 850–859, 2008.

[43] W. F. Sharpe, "The Sharpe ratio," *J. Portfolio Manage.*, vol. 21, no. 1, pp. 49–58, 1994.

[44] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, 2000.

Yue Deng received the B.E. (Hons.) degree in automatic control from Southeast University, Nanjing, China, in 2008, and the Ph.D. (Hons.) degree in control science and engineering from the Department of Automation, Tsinghua University, Beijing, China, in 2013.

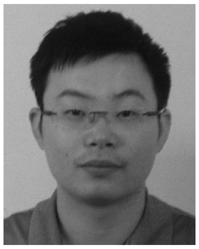
He was a Visiting Scholar with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, from 2010 to 2011. He is currently a Post-Doctoral Fellow with the School of Pharmacy, University of California at San Francisco, San Francisco, CA, USA. His current research interests include machine learning, signal processing, and computational biology.

Feng Bao received the B.E. degree in electronics and information engineering from Xidian University, Xi’an, China, in 2014. He is currently pursuing the M.S. degree with the Department of Automation, Tsinghua University, Beijing, China.

His current research interests include machine learning and signal processing.

Youyong Kong received the B.S. and M.S. degrees in computer science and engineering from Southeast University, Nanjing, China, in 2008 and 2011, respectively, and the Ph.D. degree in imaging and diagnostic radiology from the Chinese University of Hong Kong, Hong Kong, in 2014.

He is currently an Assistant Professor with the College of Computer Science and Engineering, Southeast University. His current research interests include machine learning, and medical image processing and analysis.



Zhiquan Ren received the B.S. and M.S. degrees from the School of Electrical Information and Electrical and Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2011 and 2013, respectively. He is currently pursuing the Ph.D. degree with Tsinghua University, Beijing, China.

His current research interests include machine learning, computational photography, and applications of neural network.

Qionghai Dai (SM’05) received the B.S. degree in mathematics from Shanxi Normal University, Xi’an, China, in 1987, and the M.E. and Ph.D. degrees in computer science and automation from Northeastern University, Shenyang, China, in 1994 and 1996, respectively.

He has been a Faculty Member with Tsinghua University, Beijing, China, since 1997. He is currently a Cheung Kong Professor with Tsinghua University, where he is also the Director of the Broadband Networks and Digital Media Laboratory. His current research interests include signal processing and computer vision.