Practical Deep Learning Approach for Intraday Futures Trading

Yuyuan Cui [1] **Ziyan Wang** [1]

Abstract

This paper provides the modeling approach and empirical results for using deep learning and reinforcement learning for developing systematic futures trading strategies in intraday timeframe. We build four deep neural network models and use them with various degrees of human experience intervention to formulate intraday futures trading strategies. By comparing prediction accuracy as well as trading performance against benchmark and across each other, we demonstrate that these models significantly outperform linear models and standard market making strategy, which supports applicability of deep learning to enhance intraday futures trading.

1. Introduction

Deep learning has been a popular topic in quantitative financial trading. However, most studies on high frequency trading focus on developing a single type of deep learning model. Furthermore, the asset class in question is predominantly in the equities space, and features are mostly limited to order book history of the same asset traded. In this paper, we develop our intraday strategies with a more comprehensive approach. Specifically, we build two sets of independent models. The first model is an LSTM model that focuses on short-term price change prediction only. We combine the model prediction with a manually formulated simple trading strategy to make trading decisions. The second set involves three types of deep reinforcement models (double DQN, DDPG, and SAC). The action space includes buy and sell decisions and hence these models output an entire trading strategy on its own without human intervention. Moreover, we set up linear regression based models and a passive market-making strategy as benchmark. By comparing their performance, we can evaluate the applicability of deep learning in intraday futures trading with different degrees of human experience intervention. In addition, the dataset we use is China's IH index futures tick data. This is a relatively new product that only started trading since April 2015. For feature construction, we are not constrained with order book snapshot and technical indicators, but also incorporate intraday cross-asset and time series signals that are more often seen in research on trading strategies with daily horizon.

2. Related Work

Plenty of research has been done on using cross-sectional or time series data for trading on daily horizon. Xiong, Liu, Zhong, Yang and Walid (2018) used deep reinforcement learning to obtain optimal strategy based on daily return data of 30 selected stocks (Xiong et al., 2018). Xiao Zhong and David Enke (2019) used DNNs to predict daily ETF return based on 60 financial and economic features (Zhong & Enke, 2019). Sezer, Ozbayoglua, and Dogdub (2017) used DNNs for optimizing technical indicators for daily stock trading and independently trained models for each of selected 30 stocks (Sezer et al., 2017). In intraday horizon, Prakhar Ganesh and Puneet Rakheja (2018) built a standard MLP neural network for stock price prediction based on order book snapshot and reversion signal (Ganesh & Rakheja, 2018). Svetlana Borovkova and Ioannis Tsiamas (2018) formulated stock price prediction into a classification problem and built an ensemble of LSTM networks with technical indicators as features (Borovkova & Tsiamas, 2019). Ye-Sheen Lim and Denise Gorse (2018) built Q-learning algorithm with customized CARA utility reward function and stock market order book snapshot input to optimize market making strategy (Lim & Gorse, 2018). The main distinguishing features of our research are that: we compare different types of deep learning algorithms in the same setting; we perform extensive feature engineering, incorporating microstructure snapshot, technical, time series and cross-asset features as model input; we examine the less studied futures market, in particular China's futures market.

3. Problem Formulation

The objective is to design deep learning models that maximize trading profit for IH futures[1] based on tick-level high frequency data. We will build two types of individual models in this project. The first one is a LSTM model that focuses on short-term price movement prediction only and we manually formulate a simple trading strategy based on

[1]IH future is based on SSE 50 Index, one of the most popular index futures traded in CFFEX

its predictions. The second model uses deep reinforcement learning and will output trading actions on its own. For this type, we build three individual models (double DQN, DDPG, and SAC) and will compare their performance. In addition, we will set up three benchmark trading strategies – an OLS Regression price prediction model, a Ridge Regression price prediction model, and a passive market-making strategy. By comparing our models' performance against benchmark and against each other, we can evaluate the applicability of reinforcement learning and deep learning in intraday futures trading with various degrees of human experience intervention.

4. Methods

4.1. LSTM for Mid Price Change Prediction

The reason why we use price change as our prediction target as opposed to return is that there is no fundamental change in price scale (e.g. due to share split or drastic volatility regime shift) during the entire training and testing data sample, and that our trading horizon is constrained to be intraday. In terms of architecture, we define LSTM with 1 hidden layer, 256 hidden features, and 90 samples with 90 seconds interval. Then we process the final output of LSTM with a linear model to arrive at predicted price change. The model is trained using stochastic gradient decent algorithm with time-decaying learning rate. The price change horizon we aim to predict is simple mid change in 1 second. The LSTM architecture is illustrated in Figure 1.

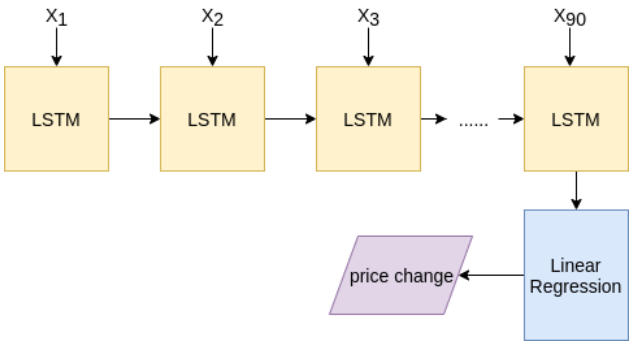

Figure 1. Architecture design of LSTM model for mid price prediction. Input of the model consists of 90 time series feature vectors $x_1, \ldots, x_{90}$, sampled with 90 seconds interval. Output of the model is fed into linear regression to produce mid price change prediction. The model is trained with stochastic gradient decent algorithm.

The time series feature vector x_t we use incorporates order book imbalance, cross-asset return, price-volume technical indicator, volatility and market microstructure signal such as bid-offer spread. More specifically, x_t is the vector of following features as of time t:

- mid_lag_T ($T = 1, 5, 10, 30$): difference between current IH mid price and its mid price T seconds before
- $wmid\_mid_T$: snapshot difference between IH weighted mid price at T and its simple mid price
- $wmid\_last\_price_T$: snapshot difference beween IH weighted mid price at T and its last traded price
- $wmid\_rolling\_min_T$ ($T = 1, 5, 10, 30$): difference of weighed mid of IH and its minimum in the past T seconds
- $wmid\_rolling\_max_T$ ($T = 1, 5, 10, 30$): difference of weighed mid of IH and its maximum in the past T seconds
- $trade\_dir_T$: most recent trade direction in IH as of time T
- mid_vol_T ($T = 30$): IH mid price volatility in the past T seconds
- $spread_T$ ($T = 10, 30, 60$): IH average bid-offer spread in the past T seconds
- $signed\_tick_T$ ($T = 10, 30$): number of up price changes minus number of down price changes in the past T seconds look-back interval
- $total\_volume_T$ ($T = 10, 30$): total IH trading volume in past T seconds look-back interval
- $signed\_volume_T$ ($T = 10, 30$): trade direction signed IH trading volume in the past T seconds look-back interval
- $IF\_mid\_lag_T$ ($T = 5, 30$): difference between current IF mid price and its mid price T seconds before
- $IF\_mid\_vol_T$ ($T = 60$): IF mid price volatility in the past T seconds
- IF_spread_T ($T = 10$): average IF bid-offer spread in the past T seconds
- $IF\_total\_volume_T$ ($T = 5, 30$): total IF trading volume in past T seconds look-back interval
- $IC\_mid\_lag_T$ ($T = 5, 30$): difference between current IC mid price and its mid price T seconds before
- $IC\_mid\_vol_T$ ($T = 60$): IC mid price volatility in the past T seconds
- IC_spread_T ($T = 10$): average IC bid-offer spread in the past T seconds
- $IC\_total\_volume_T$ ($T = 5, 30$): total IC trading volume in past T seconds look-back interval

Compared with previous research that only focus on daily returns as predictive features, we take advantage of the most granular level of tick-by-tick data (with average update frequency of less than one second), which provide a much

larger dataset for training complex deep learning models. Compared with previous high-frequency trading research that mostly used either order book snapshot or time series signal for each individual asset, we construct a much wider range of features that encompass information in order book imbalance, trading volume, trading direction, technical time series pattern, and cross-asset returns. The reason why we choose not to include cross-asset trade or book information is because based on out-sample testing, the additional predictive power to that already included in cross-asset price change is not meaningful and hence we prefer a simpler model for robustness consideration.

4.2. Deep Reinforcement Learning for Intraday Trading

The financial trading setting can be formulated as a reinforcement learning problem. The state of the agent is the amount of position it's holding, together with market condition variables x_t described in the LSTM model section. The immediate reward is just the "one-step" (2 seconds) return of its current portfolio holding, and the action space is the buy/sell decisions. As there is no ambiguity in return calculation and state transition from historical tick data, we focus our modeling effort on the optimal policy learning as opposed to modeling environment dynamics for planning. Therefore, we choose the model-free type of reinforcement learning. In the following sections, we examine three popular deep reinforcement algorithms: Q-learning based DQN, and policy optimization algorithms DDPG and SAC.

4.2.1. Double Deep Q Learning (DQN)

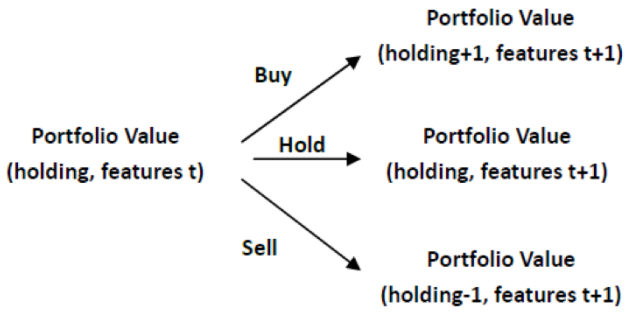

Figure 2. We've defined three actions for Q-learning: buy, hold and sell.

To characterize the state space, we will use the same features in the LSTM model for comparison purpose. In addition, we add current futures position holding as feature describing the environment. For simplicity, we restrict actions to be either long one additional contract, short one additional contract or no change in position held. As a result, the DNN's output should be three-dimensional to reflect estimate of value of these three actions. We model the Q function through a DNN with two hidden layers (each has a dimension of 10 and with ReLu activation). Furthermore, to facilitate calibration, we use market value of position holding 1 minute since initial setting up a position as "terminal state". Due to high degree of noise commonly seen in financial data, we choose a Double Q-learning approach to improve robustness and alleviate the risk of overestimating action values. Specifically, we will use the iterative approach (see Figure 3 and Algorithm 1) to update the two DNN-based $Q(\mathbf{s}_t, \mathbf{a}_t)$ function estimates, with a cyclic buffer for memory replay. The neural network parameters are optimized using the Adam algorithm.

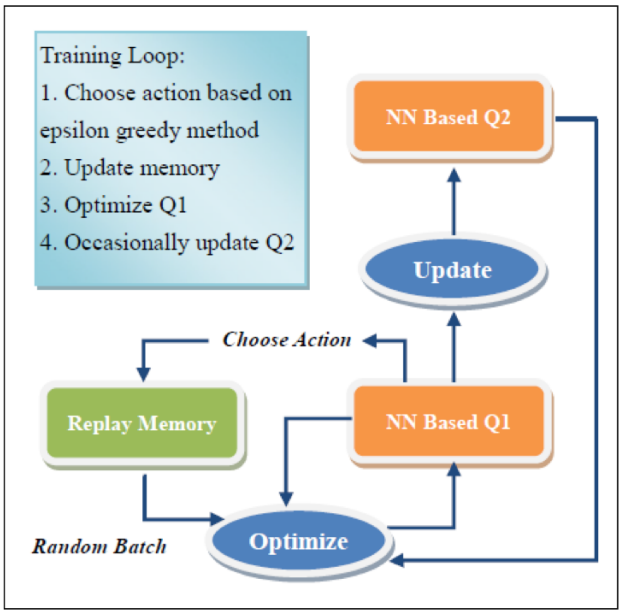

Figure 3. The training loop for the double DQN model for mid price prediction

4.2.2. Deep Deterministic Policy Gradient (DDPG)

A key limitation for the DQN framework is its discrete nature of action space. However, given the size of our trading order is variable, it is reasonable to model action as a continuous variable.

In contrast to DQN that finds optimal policy by looking up an estimated Q function, DDPG uses a DNN to directly return a single optimal action given an input state. We train the DDPG model using Actor-Critic method. The Actor is used to approximate optimal policy deterministically and is a DNN with 2 hidden layers (with dimension of 40 and 30, respectively), with ReLu activation for hidden layers and tanh for output layer. To facilitate exploration, we add an Ornstein-Ulhenbeck noise to the action produced by Actor.

$$\mu'(\mathbf{s}_t) = \mu(\mathbf{s}_t|\theta_t^\mu) + \mathcal{N}$$

Algorithm 1 Double DQN algorithm

Input: $\mathcal{D}$ - empty replay buffer; θ - initial network parameters, θ^- - copy of θ
Input: N_r - replay buffer maximum size; N_b - training batch size; N^- - target network replacement freq.
for $episodee \in \{1, 2, ..., M\}$ **do**
  Initialize frame sequence $\mathbf{x} \leftarrow ()$
  for $t \in \{0, 1, ...\}$ **do**
    Set $\mathbf{s} \leftarrow \mathbf{x}$, sample action $\mathbf{a} \sim \pi_{\mathcal{B}}$
    Sample next frame x^t from environment $\mathcal{E}$ given $(\mathbf{s}, \mathbf{a})$ and receive reward r, and append x^t to $\mathbf{x}$
    if $|\mathbf{x}| > N_f$ **then**
      Delete oldest frame $x_{t_{min}}$ from $\mathbf{x}$ end
    end if
    Set $\mathbf{s}' \leftarrow \mathbf{x}$, and add transition tuple $(\mathbf{a}, \mathbf{a}, r, \mathbf{s}')$ to $\mathcal{D}$, replacing the oldest tuple if $|\mathcal{D}| \geq N_r$
    Sample a minibatch of N_b tuples $(\mathbf{a}, \mathbf{a}, r, \mathbf{s}') \sim \text{Unif}(\mathcal{D})$
    Construct target values, one for each of the N_b tuples:
    Define $\mathbf{a}^{\max}(\mathbf{s}'; \theta = \arg\max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}'; \theta))$

$$y_j = \begin{cases} r, & \text{if } s' \text{is terminal} \\ r + \gamma Q(\mathbf{s}', \mathbf{a}^{\max}(\mathbf{s}'; \theta); \theta^-) & \text{otherwise} \end{cases}$$

    Do a gradient descent step with loss $||y_j - Q(\mathbf{s}, \mathbf{a}; \theta)||^2$
    Replace target parameters $\theta^- \leftarrow \theta \text{every} N^- \text{steps}$
  end for
end for

The Actor network is trained using sampled policy gradient:

$$J(\theta) = \mathbb{E}[Q(\mathbf{s}, \mathbf{a})|\mathbf{s} = \mathbf{s}_t, \mathbf{a}_t = \mu(\mathbf{s}_t)]$$
$$\nabla_{\theta^\mu} J(\theta) \approx \nabla_a Q(\mathbf{s}, \mathbf{a}) \nabla_{\theta^\mu} \mu(\mathbf{s}|\theta^\mu)$$
$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i [\nabla_a Q(\mathbf{s}, \mathbf{a}|\theta^Q)|_{\mathbf{s}=\mathbf{s}_i, \mathbf{a}=\mu(\mathbf{s}_i)} \nabla_{\theta^\mu} \mu(\mathbf{s}|\theta^\mu)|_{\mathbf{s}=\mathbf{s}_i}]$$

The Critic is used to approximate Q value function for a given state-action pair and is a DNN with 3 hidden layers (with dimensions of 40, 30, and 10 respectively), with ReLu activation for hidden layers and linear function for output layer. The value function is trained using the temporal difference estimate.

$$Q^{Actual}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma Q'(\mathbf{s}_{t+1}, \mu'(\mathbf{s}_{t+1}|\theta^{\mu'})|\theta^{Q'})$$
$$Loss(\theta^Q) = \frac{1}{N} \sum_i (Q^{Actual}(\mathbf{s}_i, \mathbf{a}_i) - Q(\mathbf{s}_i, \mathbf{a}_i|\theta^Q))^2$$

To improve training stability, we keep a target network for Actor (denoted as Q') and a target network for Critic (denoted as μ'). While the "regular" Actor and Critic networks are updated during the experience replay stage, the "target" networks are only updated occasionally, using a soft update strategy:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$$

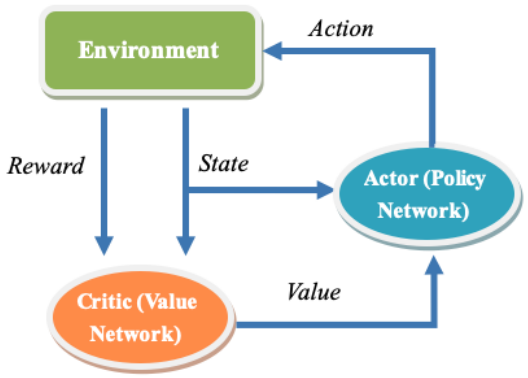

Figure 4. We train the DDPG model using Actor-Critic method where the Critic is used to approximate Q value function for a given state-action pair and is a deep neural network, and the Actor is used to approximate optimal policy deterministically.

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$$

The pesudocode is shown in Algorithm 2 and the algorithm flow chart is shown in Figure 5.

4.2.3. Soft Actor-Critic (SAC)

Another deep reinforcement model for continuous action space is the SAC model, which also considers maximizing the entropy of the policy, thus encouraging exploration and assign similar probabilities to actions with equal Q-values.

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} [r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t))]$$

where H is the entropy of policy distribution. The entropy of any continuous random variable X with probability density function p is defined as:

$$\mathcal{H}(X) = -\int_{-\infty}^{\infty} p(x) \log p(x) dx$$

We set up our SAC model with three deep neural networks:

1) a state value function V

$$\begin{aligned} V^\pi(\mathbf{s}) &= \mathop{\mathbb{E}}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \Big[\sum_{t=0}^{T} \gamma^t (r(\mathbf{s}_t, \mathbf{a}_t) \\ &\quad + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t)))|\mathbf{s}_0 = \mathbf{s} \Big] \\ &= \mathop{\mathbb{E}}_{\mathbf{a} \sim \rho_\pi} [Q^\pi(\mathbf{s}, \mathbf{a})] + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s})) \\ &= \mathop{\mathbb{E}}_{\mathbf{a} \sim \rho_\pi} [Q^\pi(\mathbf{s}, \mathbf{a}) - \alpha \log \pi(\mathbf{a}|\mathbf{s})] \\ &\approx Q^\pi(\mathbf{s}, \tilde{\mathbf{a}}) - \alpha \log \pi(\tilde{\mathbf{a}}|\mathbf{s}), \ \tilde{\mathbf{a}} \sim \pi(\cdot|\mathbf{s}) \end{aligned}$$

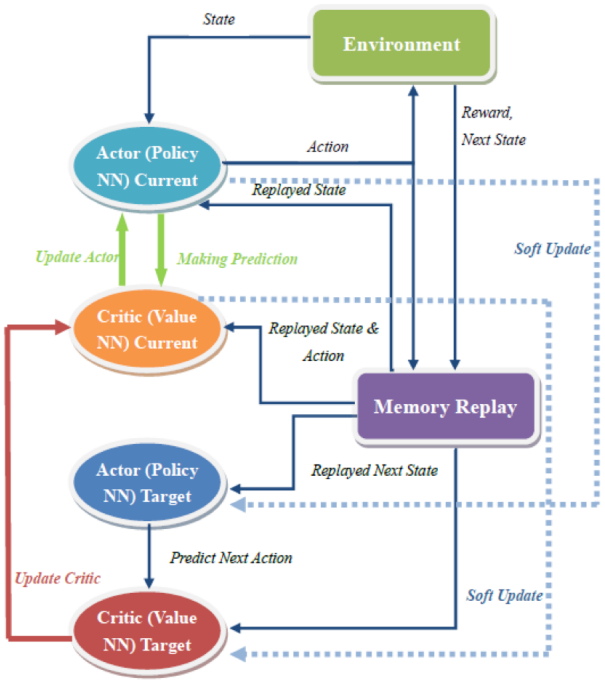

Figure 5. General overview of DDPG algorithm flow

2) a Q-function

$$
\begin{aligned}
&Q^{\pi}(\mathbf{s}, \mathbf{a}) \\
&= \mathop{\mathbb{E}}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_{\pi}} \Bigg[\sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \\
&+ \alpha \sum_{t=1}^{T} \gamma^t \mathcal{H}(\pi(\cdot|\mathbf{s}_t)))|\mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a} \Bigg] \\
&= \mathop{\mathbb{E}}_{(\mathbf{s}', \mathbf{a}') \sim \rho_{\pi}} [r(\mathbf{s}, \mathbf{a}) + \gamma(Q^{\pi}(\mathbf{s}', \mathbf{a}') + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}')))] \\
&= \mathop{\mathbb{E}}_{(\mathbf{s}') \sim \rho_{\pi}} [r(\mathbf{s}, \mathbf{a}) + \gamma V^{\pi}(\mathbf{s}')]
\end{aligned}
$$

To prevent overestimation, we use two Q networks and take the minimum value of them for both the policy and value function updates. The architecture of the two Q networks are the same - 2 hidden layers (both of dimension 30) with ReLu activation. Output activation is linear.

3) a policy function π

$$
\pi(\mathbf{a}_t|\mathbf{s}_t) \sim \mathcal{N}\Big(f_{\text{neural network}}(\mathbf{s}_t); \sum \Big)
$$

Policy is modeled through a Gaussian distribution with mean computed from a DNN that has 2 hidden layers (both have dimension 30) with ReLu activation. Output activation is linear.

We train the Q function network based on mean squared Bellman error (MSBE). To improve stability, we introduce

Algorithm 2 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with random weight θ^Q and θ^μ;
Initialize replay buffer R;
for $episode = 1, M$ **do**
  Initialize a random process N for action exploration;
  Receive initial observation state s_1;
  for $t = 1, T$ **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + N_t$ according to the current policy and exploration noise;
    Execute action a_t and observe reward r_t and state s_{t+1};
    Store transition (s_t, a_t, r_t, s_{t+1}) in R;
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R;
    Set $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{i+1}|\theta^{\mu'}|\theta^{Q'}))$;
    Update critic by minimizing the loss:

$$
L = \frac{1}{N} \sum_{i} \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2
$$

    Update the actor policy by using the sampled policy gradient:

$$
\begin{aligned}
\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_{1} &\nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \\
&\times \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}
\end{aligned}
$$

    Update the target networks:

$$
\begin{aligned}
\theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\
\theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}
\end{aligned}
$$

  end for
end for

a target network for the value function which is fixed when updating Q function. The Q function is updated using Adam algorithm.

$$
\begin{aligned}
L(\phi_i, \mathcal{D}) &= \mathop{\mathbb{E}}_{(\mathbf{s}_t, \mathbf{a}_T) \sim \mathcal{D}} [(Q_{\phi_i}(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}_1(\mathbf{s}_t, \mathbf{a}_t))^2] \\
&= \mathop{\mathbb{E}}_{(\mathbf{s}_t, \mathbf{a}_T) \sim \mathcal{D}} [(Q_{\phi_i}(\mathbf{s}_t, \mathbf{a}_t) - r(\mathbf{s}_t, \mathbf{a}_t) \\
&- \gamma \mathop{\mathbb{E}}_{\mathbf{s}_{t+1}} [V_{\psi_{targ}}(\mathbf{s}_{t+1})])^2]
\end{aligned}
$$

We train the value network based on mean-squared-error loss between current and entropy-adjusted Q value, where we use clipped double-Q form that takes the minimum of the two approximation networks. The value function is updated using Adam algorithm.

$$
\begin{aligned}
L(\psi, \mathcal{D}) = \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim \mathcal{D} \\ \mathbf{a}_t \sim \pi_\theta}} \Big[&(V_\psi(\mathbf{s}_t) - (\min_{i=1,2} Q_{\phi_i}(\mathbf{s}_t, \tilde{\mathbf{a}}) \\
&- \alpha \log \pi_\theta(\tilde{\mathbf{a}}|\mathbf{s}_t)))^2 \Big], \ \tilde{\mathbf{a}} \sim \pi(\cdot|\mathbf{s})
\end{aligned}
$$

We train the policy network using the reparameterization trick. This process ensures that sampling from the policy is

a differentiable process.

$$\tilde{a}(\mathbf{s}_t, \xi_t) = \tanh(\mu(\mathbf{s}_t) + \sigma(\mathbf{s}_t) \odot \xi_t),\ \xi_t \sim \mathcal{N}(0, I)$$

Using the first of the two Q networks as approximation for Q value, we have the optimization re-stated as the following form:

$$J(\theta) = \max_{\theta} \mathop{\mathbb{E}}_{\substack{\mathbf{s}_t \sim \mathcal{D} \\ \xi_t \sim \mathcal{N}}} [Q_{\phi_1}(\mathbf{s}_t, \tilde{\mathbf{a}}(\mathbf{s}_t, \xi_t)) - \alpha \log \pi(\tilde{\mathbf{a}}(\mathbf{s}_t, \xi_t)|\mathbf{s}_t)))^2]$$

We will update policy network with Adam algorithm.

The pseudocode for the SAC algorithm is given in Algorithm 3.

Algorithm 3 Soft Actor-Critic

Inital policy parameters θ, Q-function parameters ϕ_1, ϕ_2, V-function parameters ψ, empty replay buffer D
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer D
  if s' is terminal **then**
    reset environment state
  end if
  if If it's time to update **then**
    for j in range (however many updates) **do**
      Randomly sample a batch of transitions, $B = (s, a, r, s', d)$ from D
      Compute targets for Q and V functions:

$$y_q(r, s', d) = r + \gamma(1 - d)V_{\psi_{targ}}(s')$$

$$y_v(s) = \min_{i=1,2} Q_{\psi_i}(s, \bar{a}) - \alpha \log \pi_\theta(\bar{a}|s), \quad \bar{a} \sim \pi_\theta(\cdot|s)$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi,i}(s, a) - y_q(r, s', d))^2$$

      for $i = 1, 2$
      Update V-function by one step of gradient descent using

$$\nabla_\psi \frac{1}{|B|} \sum_{s \in B} (V_\psi(s) - y_v(s))^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} (Q_{\phi,1}(s, \tilde{a}_\theta) \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s)$$

      where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is a differentiable wrt θ via the reparametrization trick
      Update target value network with

$$\psi_{targ} \leftarrow \rho\psi_{targ} + (1 - \rho)\psi$$

    end for
  end if
until convergence

5. Results

5.1. Data

The dataset we use is tick data of IH futures (SSE 50 Stock Index Futures), IC futures (CSI 500 Stock Index Futures) and IF (CSI 300 Stock Index Futures) futures tick data in China's CFFEX exchange in 2018. The data include all book updates and trade events. The morning trading session is 09:30 AM - 11:30 AM and the after session is 01:00 PM – 03:00 PM. We treat data in any look-back period outside of session (e.g. 9:20 AM or 12:30 PM) as NaN but otherwise treat both sessions equally. For each futures, we use front-month contract (contract that expires in the next month). As maximum tick frequency is around 0.5 seconds, we regularize all data with resample frequency of 0.5 seconds. For missing data, we fill last traded price, bid/offer price and size with most recent valid value; and fill traded volume with 0. In back-testing, we always close all positions whenever a trading session ends. The model training (in-sample) period is 1/1/2018 to 8/31/2018, validation period is 9/1/2018 to 9/30/2018 and the testing (out-sample) period is 10/1/2018 to 12/31/2018.

5.2. Model Performance

5.2.1. LSTM for Mid Price Change Prediction

We evaluate the LSTM model using mean-squared error (MSE) of predicted compared with actual mid price change. Model performance is compared with OLS and Ridge Regression using same set of features as benchmark. Table 1 shows that LSTM has significantly lower MSE and the generalization error (measured by increase from in-sample to out-sample MSE) is also lower. Therefore, from fitting perspective, the predictive power of LSTM is better and more robust than standard linear models.

	In-sample MSE	Out-sample MSE
LSTM	0.56	0.75
OLS	1.70	2.22
Ridge Regression	1.72	2.15

Table 1. Performance comparison between our LSTM model and ridge regression model and OLS model

Next, we examine its performance through two-way table. As Table 2 and Table 3 show, in a predominant portion of sample, the short-term mid change is smaller than one tick (a full bid-ask spread), which means that even with perfect predictive modeling, we may still not be able to trade profitably after accounting for the transaction cost of crossing the bid-ask spread. A good trading signal should capture market moves that are more than one tick and avoid taking unnecessary positions that won't realize gains in

excess of transaction cost. LSTM achieves this objective better than linear models. In Table 3, in out-sample trading, LSTM makes less error in taking a position bet that turns out to be a move within a tick and makes more correct bet when the market actually moves more than a tick. In contrast, linear models make more conservative or wrong prediction when market actually makes major move. Next,

Actual \ Predicted	< -1 Tick	[-1 Tick, +1 Tick]	> +1 Tick
< -1 Tick	0%	8%	3%
[-1 Tick, +1 Tick]	1%	61%	10%
> +1 Tick	2%	10%	5%

OLS

Actual \ Predicted	< -1 Tick	[-1 Tick, +1 Tick]	> +1 Tick
< -1 Tick	1%	9%	1%
[-1 Tick, +1 Tick]	3%	62%	7%
> +1 Tick	2%	12%	3%

Ridge

Actual \ Predicted	< -1 Tick	[-1 Tick, +1 Tick]	> +1 Tick
< -1 Tick	5%	5%	1%
[-1 Tick, +1 Tick]	2%	65%	5%
> +1 Tick	0%	8%	9%

LSTM

Table 2. Two-way table of in-sample training

we compare the performance of a simple trading strategy guided by the LSTM with the benchmark. The LSTM-based strategy buys when it predicts the price will move up by at least 0.25, places sells when it predicts the price will move down by at least 0.25, and takes no action otherwise. The benchmark strategy, however, always places both a bid and an offer order. We restrict both strategies to hold a maximum of three contracts. As Figure 7 shows, the LSTM model outperforms the benchmark strategy while linear models have a hard time making profits.

5.2.2. Deep Reinforcement Learning for Intraday Trading

Figure 6 shows the average of total rewards in the rolling 10 epochs of training. All three reinforcement learning models achieves a stable level within 1,000 epochs. To better evaluate them in a trading strategy setup, we look at average immediate reward of taking an action, which is measured as one-step (2 seconds) return of position holding after taking action. All three models outperform the benchmark strategy. Furthermore, DDPG and SAC has higher rewards in the "Buy" and "Sell" action columns. This is because their action space is continuous and that allows the models to bet more aggressively in times of greater confidence.

Actual \ Predicted	< -1 Tick	[-1 Tick, +1 Tick]	> +1 Tick
< -1 Tick	1%	11%	1%
[-1 Tick, +1 Tick]	7%	51%	11%
> +1 Tick	2%	15%	1%

OLS

Actual \ Predicted	< -1 Tick	[-1 Tick, +1 Tick]	> +1 Tick
< -1 Tick	0%	12%	1%
[-1 Tick, +1 Tick]	4%	57%	8%
> +1 Tick	1%	16%	1%

Ridge

Actual \ Predicted	< -1 Tick	[-1 Tick, +1 Tick]	> +1 Tick
< -1 Tick	2%	10%	1%
[-1 Tick, +1 Tick]	4%	61%	4%
> +1 Tick	1%	12%	5%

LSTM

Table 3. Two-way table of out-sample testing

	Avg Sell Reward	Avg Hold Reward	Avg Buy Reward
MM	0.19	0.08	0.16
DQN	1.68	0.85	1.65
DDPG	5.82	1.47	3.77
SAC	7.84	1.71	10.10

	Avg Sell Reward	Avg Hold Reward	Avg Buy Reward
MM	0.21	0.08	0.17
DQN	0.86	0.64	0.70
DDPG	3.57	0.47	2.93
SAC	3.42	0.54	4.65

Table 4. Comparison of average rewards of buy, hold and sell actions for three reinforcement algorithms: DQN, DDPG and SAC, and a market making strategy

5.2.3. Trading Performance Comparison

We put the cumulative profit-and-loss (pnl) plot for all of the models together for out-sample backtesting in Figure 7. In addition, we calculate trading performance statistics for these models in Table 5. In terms of overall return, DDPG is the best, followed by SAC. This shows that the flexibility of position sizing in these algorithms brings greater return potential. However, it does not appear entropy regularization in SAC results in meaningful improvement from DDPG. In terms of risk profile, LSTM and DQN have a lower Sharpe but also lower drawdown and volatility. This makes them more suitable for risk-averse investors. The four deep learning models all beat the benchmark strategies.

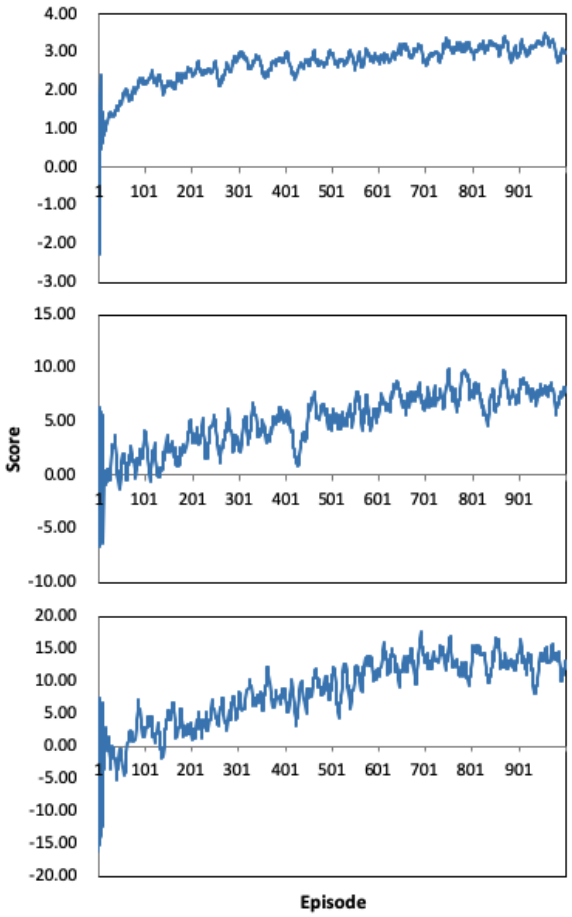

Figure 6. Comparison of in-sample learning performance (measured by moving average of returns over most recent 10 epochs of training) for the three deep RL models. From top to bottom are for DQN, DDPG and SAC respectively. Out-sample backtesting period is 10/01/2018 – 12/31/2018

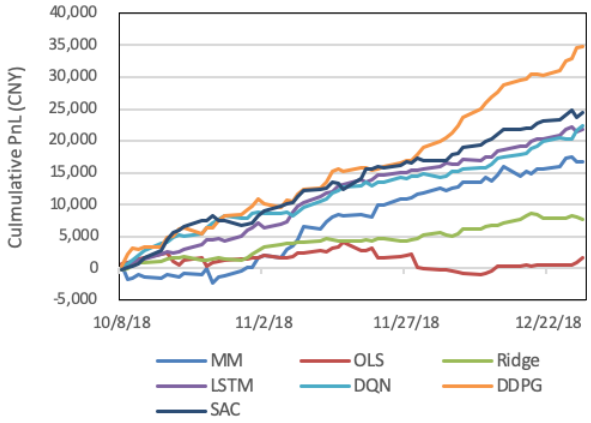

Figure 7. Comparison of cumulative trading PnL for each of the 7 trading strategies (strategies based on OLS, Ridge, LSTM, DQN, DDPG, SAC, and the benchmark market making strategy) over the out-sample backtesting period (10/01/2018 – 12/31/2018)

	MM	OLS	Ridge	LSTM	DQN	DDPG	SAC
Avg Daily PnL	292	133	169	475	515	741	848
Std Dev Daily PnL	617	490	271	288	310	416	557
Sharpe	0.47	0.27	0.62	1.65	1.66	1.78	1.52
Max Drawdown	2,759	1,587	606	305	165	517	244
Turnover	3,319	3,608	3,032	2,458	2,107	7,251	6,114

	MM	OLS	Ridge	LSTM	DQN	DDPG	SAC
Avg Daily PnL	278	27	129	365	374	580	407
Std Dev Daily PnL	782	557	298	403	400	631	589
Sharpe	0.36	0.05	0.43	0.90	0.93	0.92	0.69
Max Drawdown	2,832	4,942	831	795	535	1,411	1,619
Turnover	3,903	3,321	3,265	2,662	2,443	7,632	6,589

Table 5. Trading performance statistics for for each of the 7 trading strategies (strategies based on OLS, Ridge, LSTM, DQN, DDPG, SAC, and the benchmark market making strategy). In-sample data is shown in the upper table while out-sample data is shown in the lower table. In-sample period is 01/01/2018 – 8/31/2018, and out-sample period is 10/01/2018 – 12/31/2018

5.3. Analysis

The LSTM model has greater flexibility in capturing both the sequential dependency and complex inter-feature interaction of the high dimensional feature time series. As a result, both in-sample and out-sample MSE are significantly lower than OLS and Ridge. The model-based strategy also outperforms passive (uninformed) trading strategy. The benchmark strategy trades more frequently and often suffers from adverse selection. In comparison, both the LSTM model and the three RL models are more cautious in entering into positions and only trade when they have confidence in predicted direction. Extending LSTM model with human input to end-to-end deep reinforcement learning (DQN, DDPG, and SAC), we achieve even higher return This means that we can delegate both price prediction and position management to deep learning models.

5.4. Project Repository

https://github.com/louiswang01/deep-learning-for-futures-trading

References

Borovkova, S. and Tsiamas, I. An ensemble of lstm neural networks for high-frequency stock market classification. *Journal of Forecasting*, 38(6):600–619, 2019. doi: 10.1002/for.2585. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/for.2585.

Ganesh, P. and Rakheja, P. Deep neural networks in high frequency trading, 2018.

Lim, Y. and Gorse, D. Reinforcement learning for

high-frequency market making. In *26th European Symposium on Artificial Neural Networks, ESANN 2018, Bruges, Belgium, April 25-27, 2018*, 2018. URL http://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2018-50.pdf.

Sezer, O. B., Ozbayoglu, M., and Dogdu, E. A deep neural-network based stock trading system based on evolutionary optimized technical analysis parameters. *Procedia Computer Science*, 114:473 – 480, 2017. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2017.09.031. URL http://www.sciencedirect.com/science/article/pii/S1877050917318252. Complex Adaptive Systems Conference with Theme: Engineering Cyber Physical Systems, CAS October 30 – November 1, 2017, Chicago, Illinois, USA.

Xiong, Z., Liu, X.-Y., Zhong, S., Yang, H., and Walid, A. Practical deep reinforcement learning approach for stock trading, 2018.

Zhong, X. and Enke, D. Predicting the daily return direction of the stock market using hybrid machine learning algorithms. *Financial Innovation*, 5 (1):24, Jun 2019. ISSN 2199-4730. doi: 10.1186/s40854-019-0138-0. URL https://doi.org/10.1186/s40854-019-0138-0.