Reinforcement Learning for FX trading

Yuqin Dai, Chris Wang, Iris Wang, Yilun Xu
Stanford University
{alexadai, chrwang, iriswang, ylxu} @ stanford.edu

1 Introduction

Reinforcement learning (RL) is a branch of machine learning in which an agent learns to act within a certain environment in order to maximize its total reward, which is defined in relationship to the actions it takes. Traditionally, reinforcement learning has been applied to the playing of several Atari games, but more recently, more applications of reinforcement learning have come up. Particularly, in finance, several trading challenges can be formulated as a game in which an agent can be designed to maximize a reward. Reinforcement learning presents a unique opportunity to model the complexities of trading in which traditional supervised learning models may not be able to explore. FX trading is one such financial problem.

FX trading involves trading currency pairs in a large, decentralized market, using various brokers to trade. Each currency pair represents the value of the first currency relative to the second one. Each broker will usually quote both a bid price and ask price, where the bid price is the amount the broker is willing to purchase the pair from you for, and the ask price is the price you need to pay the broker to give you a share of the pair. Because of the complexities of this market and transaction structure, we decided to apply reinforcement learning to this problem.

2 Background and related work

2.1 Classical FX trading

In foreign exchange markets, currency pairs are traded in such way that a quoted price is used. The average of FX daily trading volume is more than $5.1 trillion. [5] Factors like interest rates, trade flows, tourism, economic strength and geopolitical risk affect supply and demand for currencies, creating volatility in the FX market. Currency as an asset class has two distinct features, making it an ideal asset to trade: different interest rate in different countries, and fluctuation in the exchange rate.

The FX market is the largest marketplace in terms of daily trading volume and therefore offers the most liquidity, which means the spread is relatively smaller and the barrier to enter or exit a position is also smaller. Meanwhile, trading currencies can be risky and complex. People use various strategies to trade in the FX market, for example, statistical or algorithmic execution. [4]

2.2 Reinforcement learning

Reinforcement Learning is a type of machine learning technique that can enable an agent to learn in an interactive environment by trials and errors using feedback from its own actions and experiences, as shown in figure 1.

Compared to other machine learning techniques, reinforcement learning has some unique characteristics. Both supervised learning and reinforcement learning are mapping from input to output. However, unlike supervised learning for which the feedback is the correct labels (classification) or correct output (regression), reinforcement learning interact with the environment and get feedback through rewards. Reinforcement

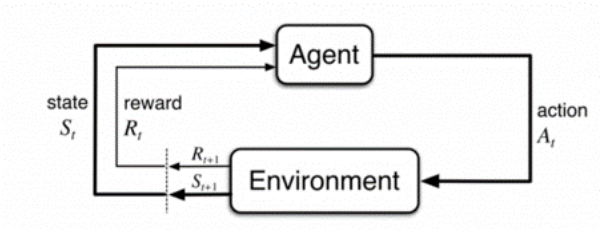


Figure 1: Reinforcement Learning Model

learning also has different learning goals from supervised and unsupervised learning. The latter two learn to make the best predictions, while reinforcement learning learns to pick actions that would maximize the long-term cumulative reward, which resembles the goal of real-world trading.

Reinforcement learning can be further categorized into model-based and model-free algorithms based on whether the rewards and probabilities for each step are readily accessible, which in most cases are not available. Policy gradient and deep Q-network are two commonly used model-free algorithms. Policy gradient is a policy-based approach that directly parameterize the policy [9]. This method has several advantages such as better convergence properties and ability to cover continuous action spaces. While deep Q-network is a value-based method which combines deep learning with Q-learning, with the learning objective to optimize the estimates of action-value function [6].

2.3 Reinforcement learning in financial market

Reinforcement learning has been an area of interest for both academia and industry. In particular, it has been widely applied to develop investment and trading strategies in financial market.

Nevmyvaka et al. [8] introduces an efficient RL algorithm that fuses Q-learning and dynamic programming and exploits the approximate independence between private and market variables. The paper experiments the algorithm on NASDAQ stocks and achieves in execution of 50% or more over submit-and-leave policies.

Moody and Saffell [7] proposes an direct reinforcement learning algorithm for discovering investment strategies, which eliminates the need to build forecasting models. Specifically, the paper infuses a recurrent architecture into reinforcement learning and applies it on USDGBP FX trading and S&P 500/treasury bills asset allocation, the results of which outperform systems trained by supervised methods. Deng et al. [1] uses a deep direct RL network with fuzzy representations to reduce uncertainties in data. The paper tests the trading model on both stock index and commodity futures contracts and compares the performance with prediction-based DNNs. The results show that the RL model achieves stable profits taking into account trading costs.

Huang [3] proposes an Markov Decision Process (MDP) model for the financial trading and solve it with the deep recurrent Q-network (DRQN) algorithm using a substantially small replay memory. The paper validates the algorithm on the spot foreign exchange market and achieves positive results under most simulation settings.

3 Data

We use a high-frequency FX trading dataset provided by Intergal. The dataset contains millisecond-level bid-ask offerings from 5 major liquidity providers for 8 currencies (AUDUSD, EURUSD, GBPUSD, NZDUSD, USDJPY, USDSEK, USDCAD and USDCHF).

Table 1: Example of raw data

Time	Currency	LP	Bid price	Ask price	Bid quota	Ask quota
20190201 00:00:00.003	AUDUSD	LP-1	0.7214	0.7218	1,000,000	1,000,000

The dataset provides key order information from 2019/02/01 to 2019/03/01. At each time step, our strategy receives no more than 5 new bid/ ask offers from the liquidity providers, each offer containing the bid/ ask prices together with allowed transaction quotas. Most bid/ ask offers in our dataset are associated with the quota of 1,000,000 local currency. As a result, we constrain our trading scale under 1,000,000 to ensure that we use valid offers for training and evaluation of our strategy.

3.1 Data processing

To begin with, we clean all invalid records and down sample the dataset into a second-level one. Then, we pad the data to the same timeframe so that all currency pairs have 5 valid bid/ask offers from the liquidity providers at each time step. Furthermore, we build an order book to select the best bid and ask offers at each time step. As shown in figure 2, we rank all the bid/ ask offers and select the the lowest ask price and the highest bid price, which in turn narrows down the bid-ask spread. Moving forward, we use the best offers to build and evaluate our strategy.

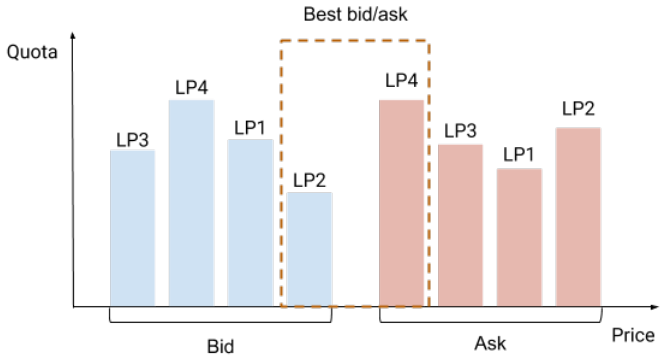


Figure 2: Order Book

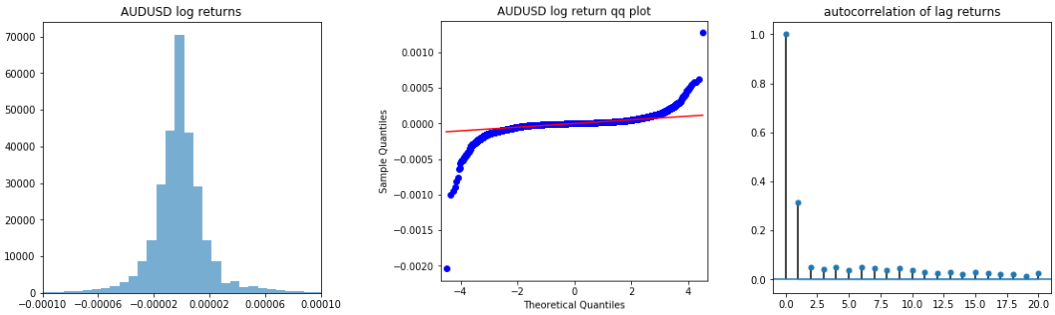
3.2 Data visualization

Before diving into our models, we analyze the dataset to understand the statistical characters of the high frequency data. As shown in figure 3, 1-second log returns of AUDUSD distribute tightly around 0, and exhibit strong long-tail property. Compared to daily data, second-level FX returns fails to show significant auto-correlation. As a result, we decide to adopt no prior distribution in our models.

Meanwhile, we exam the relationship between different currencies. The result shows that several currency pairs, e.g. AUDUSD - NZDUSD, EURUSD - GBPUSD, have strong correlation between each other. Thus, we extract features from all currency pairs in our models.

3.3 Training pipeline

For this project, we focus on trading AUDUSD, EURUSD and GBPUSD. For each currency pair, we split the full-month data into 5 trading weeks. We use the first 3 weeks to train 3 separate models, and evaluate/ test their performance on the first two days of the following week. To avoid overfitting, we design and tune our model structures only on AUDUSD. Thus, the results from EURUSD and GBPUSD can better represent unbiased performance of our strategy.



(c)

Figure 3: Statistical features

	2/1	2/3	2/4	2/5	2/6	Train Week 1
Eval/test Week 1	2/7	2/8	2/10	2/11	2/12	Train Week 2
Eval/test Week 2	2/13	2/14	2/15	2/17	2/18	Train Week 3
Eval/test Week 3	2/19	2/20	2/21	2/22	2/24	
	2/25	2/26	2/27	2/28	3/1	

Figure 4: Training pipeline

4 Direct reinforcement learning

4.1 Problem formulation

The goal is to take the time series data, with optimal historical bid and ask spreads, and make a trading decision based on this known information. Specifically, at a given time step, we have the historical bid and ask spreads for each of 8 currencies, and we want to make a decision on how we should trade one of the 8 currencies specifically. While our end goal is to be able to make decisions on a universal time scale, in order to apply a reinforcement learning approach to this problem with rewards that do not occur at each step, we formulate the problem with a series of episodes. In each episode, which we designate to be one hour long, the agent will learn the make decisions to maximize the reward (return) in that episode, given the time series features we have.

4.2 Method and motivation

Performance of an investment depends on both the current actions as well as their relationship to future actions. This relationship naturally leads us to reinforcement learning. Based on the learning goals, most reinforcement learning algorithms can be bucketed into critic-based and actor-based methods. Critic-based methods, such as Q-learning or TD-learning, aim to learn to learn an optimal value-function for a particular problem. The value-function of a state will include the future expected reward, and requires iterating with dynamic programming to calculate. In contrast, actor-based methods only necessitate an objective function which determines the action at each step directly from the information given. Rather than using discrete states and values to find the best state to go to, the direct actor-based model can infer the policy directly from the given features. Thus, we first attempt a direct actor reinforcement learning model on our dataset.

At each time step, we define the feature vector consisting of the bid and ask spreads of the 8 other currencies over the last constant number of time steps as f. We also take into account the previous action a. We predict the next action using a three layer neural network $g(f)$, with the equation $a_{new} = tanh(g(f) + ua + b)$, where u, b are additional learnable parameters. The $tanh$ function maps the output of the equation to a float

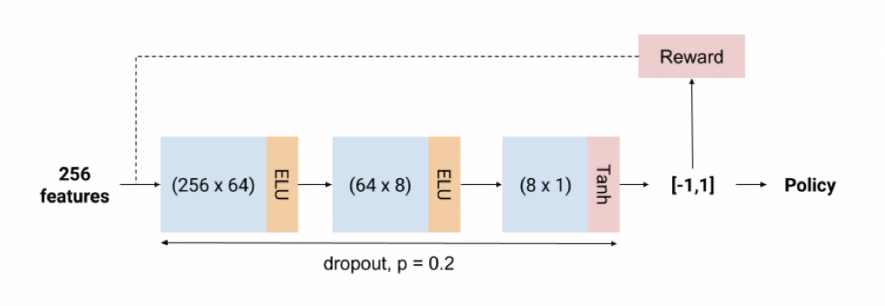


Figure 5: Deep Reinforcement Learning

value between -1 and 1, which represents the action that we take at the next step, which is a continuous value between -1 (short will all cash) and 1 (long with all cash).

Algorithm 1: Deep reinforcement learning

Initialize: Differentiable policy parameterization $\pi(a|s,\theta)$ (i.e., trading agent)
for $l \leftarrow 0$ **to** L **do**
  Generate a new episode $(s_0, a_0, r_1, ..., a_{\tau-1}, r_\tau)$ following current $\pi(a|s,\theta)$
  for $t \leftarrow 0$ **to** $\tau = 3,600$ **do**
    Cumulative return $G \leftarrow$ return from step t (G_t)
    $\theta \leftarrow \theta + \alpha\gamma^t G \nabla_\theta \ln \pi(a_t|s_t,\theta)$
    $t = t + 1$
  end
end

We train the weights of this equation by backpropogating the total rewards across the 3600 second time steps during the entire hour. To calculate these rewards, we design a reward function that represents a realistic depiction of the amount of money that can be expected to be made if the predicted actions were carried out, and represented the cash flow. Any amount of the currency pair that is bought represents a reward of $r = -(\textit{Ask Price})(\textit{Units Purchased})$, while any amount sold represents a reward of $r = (\textit{Bid Price})(\textit{Units Sold})$. Note this amount is actually based on the difference between the previous and current actions. These amounts account for the spread and are expressed as if any amount in the currency pair is not cash and the reward is relative to solely our cash holdings. We make sure that at the end of the hour, the position is liquidated, so that a realistic return is calculated. The total reward is thus the sum of the reward of all the time steps $R = r_1 + r_2 + ... + r_{3600}$. This reward is exactly what one would expect to make trading using the posted quotes and prices. At the end of the step, the reward is back-propagated to the weights of the network and equation.

4.3 Experiments

We first conduct a series of preliminary experiments to set hyper-parameters before training the model on particular currency pairs. The main parameters we attempted to analyze were optimizer, number of features for input, the presence of bias, the number of hidden layers, and whether to use dropout. In each experiment, we ablated one of these variables at a time and kept the other variables constant, and trained for 50 epochs, before comparing performance on the testing set.

Overall, we were able to find that SGD with 256 features (16 time step history), bias, a 64x8 hidden layer structure, and dropout were the most effective. We hypothesize that SGD is more effective than Adam even though it starts slower because adaptive networks might have a tendency to overfit to certain properties on the training set. A lower number of time steps actually performing better can be explained by the fact that a lot of the time-steps might contain noisy data (as we will se in the feature maps in part 6), which prevents the

model from converging easily. Dropout prevents the model from overfitting, and the bias and ideal number of hidden layers give the model more opportunities to develop an accurate representation in this case. Using these hyperparameters, we then train our model on three different currency pairs (AUDUSD, EURUSD, GBPUSD) for three different weeks, using the first four days of the week as the training set, the fifth day as the eval/val set, and the sixth day as the test set. The results are described in part 6.1.

5 Deep Q-Network

5.1 Problem formulation

Direct reinforcement learning has established a framework in which the agent can take a series of bid/ask prices and generate continuous trading decisions between -1 and 1. However, we may hope to define a legal action set with finite number of trading decisions for execution purpose. Meanwhile, the direct reinforcement learning model is optimized to maximize hourly trading returns. In certain real-life situations, we may prefer to stay for a shorter or longer period in the market, while it doesn't make sense to retrain the model every time. Given the two potential drawbacks of direct reinforcement learning model, we consider experimenting on deep Q-network (DQN).

5.2 Method and motivation

Similar to direct reinforcement learning, we adopt the episodic training method for DQN. At each iteration, we draw a 1-hour episode from the data. At each time step, the DQN agent processes the feature through a deep neural network and outputs a legal action within $\{-1, 0, 1\}$, where -1/ 1 stands for building a full short/ long position with all cash, and 0 indicates taking neutral position. At each second, we construct the feature span from the following 3 sources:

- Market feature $\in \mathcal{R}^{144}$: We compute 16-second log returns of the target currency's bid/ ask prices and other currency pairs' mid prices. We intend to use the 16-second price movement to describe the market condition.
- Position feature $\in \mathcal{R}^3$: We encode the previous position into a one-hot vector so that the model may learn to avoid unnecessary transactions and perform decision making across different time steps.
- Time feature $\in \mathcal{R}^4$: We also encode the day of the week, hour, minute and second of each timestamp into 4 features, so that the model may learn to capture the different trading cycles.

The DQN algorithm is based on the concept of optimal state-action values Q^*. In brief, we assume the trading environment follows a Markov Decision Process. Applying Bellman's equation, we may write the value function of each state-action pair as:

$$Q^*(s,a) = r + \gamma \sum_{s' \in \mathbf{S}} P^a_{ss'} \max_{a' \in A} Q^*(s', a')$$

Where r is the observed return and γ is the discount factor. Given that the financial market have infinite potential states, we use a deep neural network with the structure of figure 6 as function approximator to represent the optimal Q^* value for each state-action pair.

In order to learn the Q^* efficiently, DQN adopts a incremental TD method which can be written as follows:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,s',r)} \left[||r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s,a)||^2 \right] \tag{1}$$

$$\theta = \theta - \alpha \nabla_\theta \mathcal{L}(\theta) \tag{2}$$

During the training process, we generate a series of episodes from the trading environment and store them into a buffer. Then, we draw batches from the buffer and update our optimal Q^* in parallel. The process can be shown in the following snippet.

Algorithm 2: DQN with action augmentation

Initialize: Q-network Q_θ and target network Q_{θ^-}, where $\theta^- = \theta$
Environment $\mathcal{E}$ based on trading data $\mathcal{D}$; Buffer $\mathcal{B}$
for $l \leftarrow 0$ **to** L **do**
  Reset the environment to receive starting state s_0 for episode l
  for $t \leftarrow 0$ **to** $\tau = 3,600$ **do**
    $t = t + 1$
    Select greedy action w.r.t. $Q_\theta(s, a)$ and apply to environment $\mathcal{E}$
    Receive reward r and next state s'
    Augment actions and store the new experience to memory $\mathcal{B}$
  end
  Update θ^- with θ periodically
  Update DQN with $\mathcal{B}$ using (1) and (2)
end

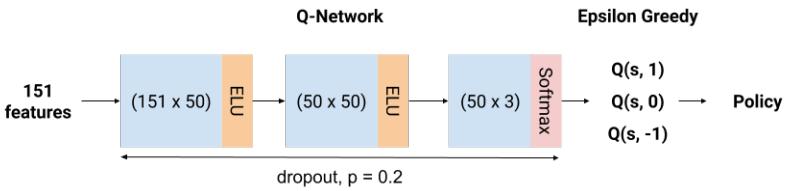


Figure 6: Deep Q-network

Meanwhile, we customize the Q-network in a few ways to make it work better in a financial trading environment. To begin with, we build our own environment so that the agent can visit the states (bid/ask prices) sequentially, without leaking the future information. Then, we tune the buffer size so that the Q-network can be trained on most recent data. Also, we use action augmentation proposed by Huang [3] to encourage exploration. At each step, our model will choose an action, e.g., short, with reward 10. Then we will naturally know that the reward for long is -10 while for neutral is 0. We add all those three actions to the buffer, which will in turn encourage deep exploration.

5.3 Experiments

As we will detail in section 6.2, we first train a model using trading data from week 1 on AUDUSD. A set of hyper-parameters, including different optimizers, number of features, hidden layers, and dropout, are tuned on the dataset. However, none of the combinations manages to achieve reasonable performance on the training and evaluation dataset.

6 Simulation and discussion

6.1 Direct reinforcement learning

For each currency pair, we separate them by week and train a model for each week. For the three currency pairs, namely AUDUSD, EURUSD, GBPUSD, over the three weeks, we train altogether 9 models. We split the weekly data to train, validation and test and the results are shown below.

During training, we evaluate the model every 50 episodes using 25-hour data. We use google cloud to train the model. We train the model for 150 epochs, each epoch containing 25 episodes. It takes 12 hours to train a full model and the rewards on evaluation set per training are shown in the figures below.

We use the test data set, i.e. 48 hours on the following week, to test the model.

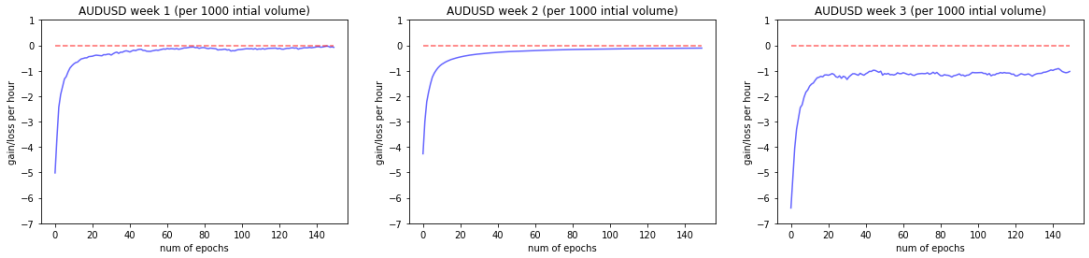


Figure 7: AUDUSD evaluation during training time

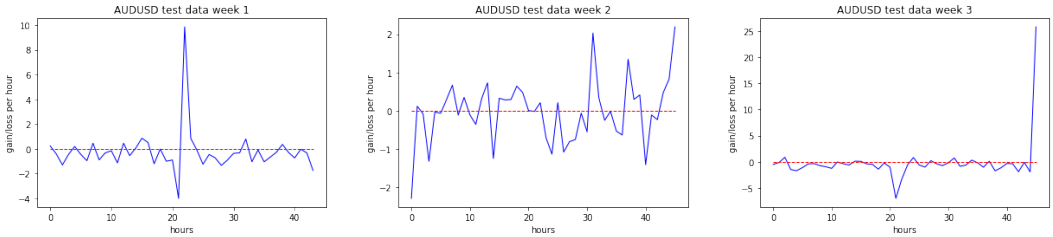


Figure 8: AUDUSD model performance on test set

Table 2: AUDUSD model performance summary on test data set

	week 1	week 2	week 3
Mean Return ($)	-0.25	0	-0.12
Variance ($)	1.745	0.811	4.06
Yield	-0.036%	0%	-0.017%

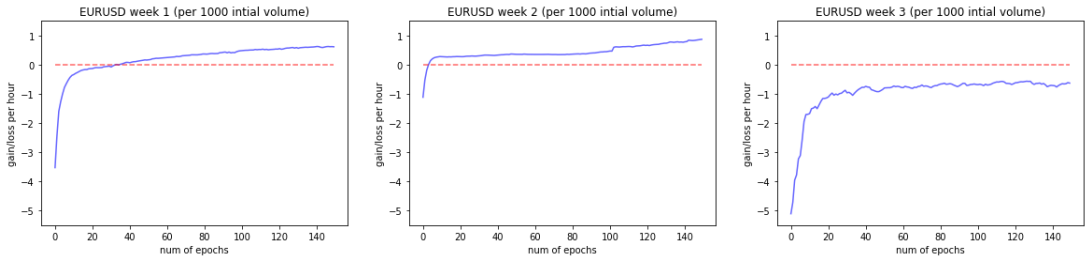


Figure 9: EURUSD evaluation during training time

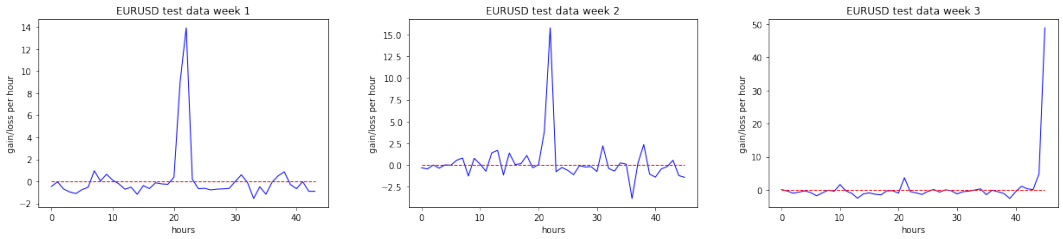


Figure 10: EURUSD model performance on test set

Table 3: EURUSD model performance summary on test data set

	week 1	week 2	week 3
Mean Return ($)	0.18	0.30	0.69
Variance ($)	2.758	2.593	7.291
Yield	0.014%	0.023%	0.052%

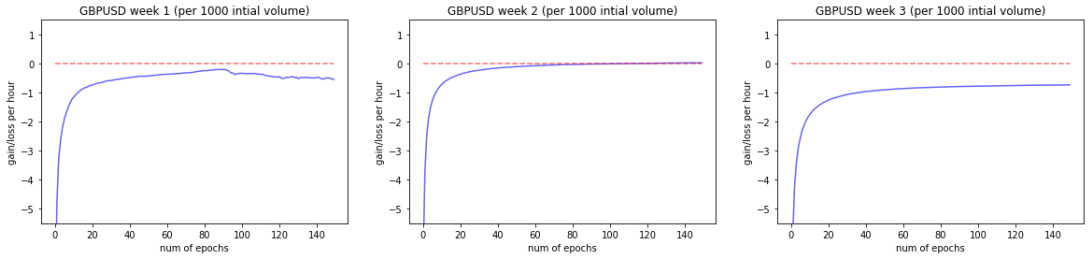


Figure 11: GBPUSD evaluation during training time

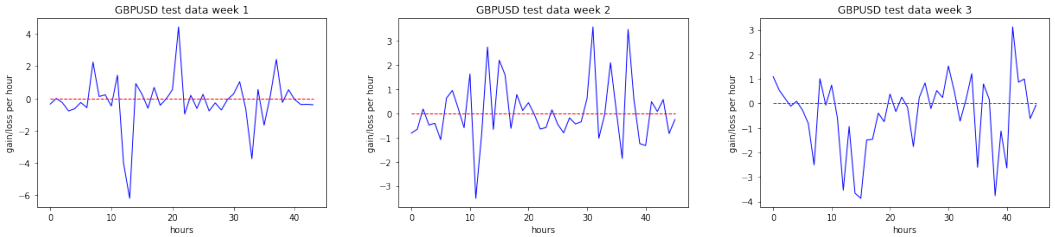


Figure 12: GBPUSD model performance on test set

Table 4: GBPUSD model performance summary on test data set

	week 1	week 2	week 3
Mean Return ($)	-0.21	0.072	-0.413
Variance ($)	1.589	1.298	1.478
Yield	-0.016%	0.0056%	-0.032%

Most models manage to break even on the evaluation/test set, while the ones on EURUSD can generate positive returns. It is also noted that for EURUSD currency pair, the model is performing slightly better on test set, which signifies that the model is not overfitting. The spike of profit can be noticed in AUDUSD and EURUSD. The reason behind the spike could be that the model is conservative in transaction, since the bid-ask spread

will always result in loss, and frequent transactions are trained to be avoided. Therefore small profit and loss are more common in the results. It is also useful to bear in mind that the way we calculate reward are based on cash flow, not the fair value of the currency pair.

Furthermore, we simulate the portfolio growth under our strategy. In the simulation, we initialize the model with 1,000 EUR, which is equivalent to 1,130 USD, using quotes from February 2019. Then we make transaction per second based on the decision made by the model. We constantly trade EURUSD for 48 hours and our cumulative return could be shown as follows.

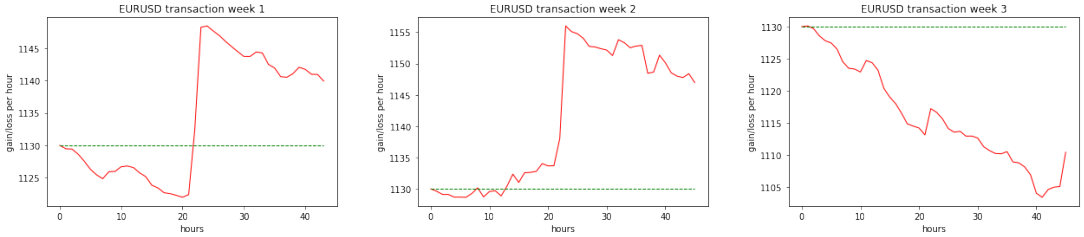


Figure 13: EUR-USD model consistent trading strategy simulation

Although the neural-network model achieves a relatively high predictive performance, the strategy is hard to visualize/ conceptualize. We assess the relative input feature importance by numerically calculating the partial differentiation of the action with respect to the input features [2]. By calculating the sensitivity of the input features, we could intuitively quantify the model and understand the input feature importance. The first 32 input features and their relative importance are plotted in figure 14.

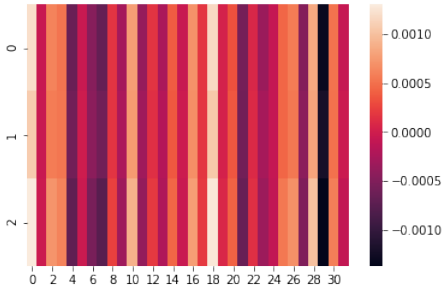


Figure 14: Partial differential of input features

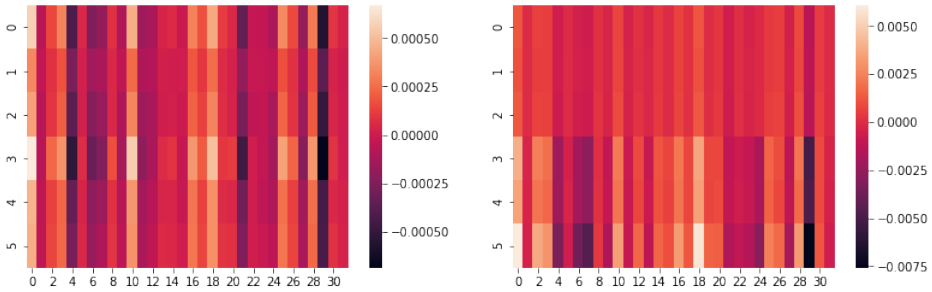


Figure 15: Partial differential of input features across time and currency

The left figure of figure 15 shows the derivatives of GBPUSD models trained on three different weeks. The result indicates that the significance of features are consistent across time, which suggests that the hidden structure keeps looking for similar patterns across time.

However, the derivatives appear to be different when we compare the sensitivity of input features across different currency. The right figure of figure 15 shows relative importance of features for GBPUSD (top 3) and EURUSD (bottom 3), indicating that each currency pair has its own patterns developed during trading.

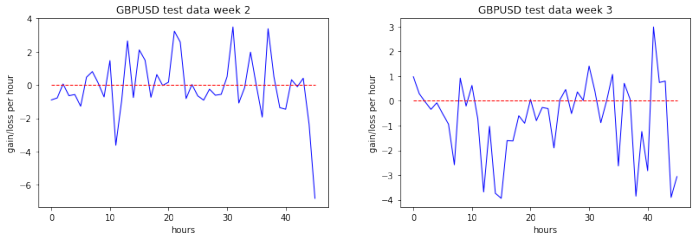


Figure 16: GBP-USD model with time lag

Figure 16 depicts the change of market dynamics. It can be seen that as the time lag increases, the model performance deteriorates. Therefore, it demonstrates that the market is constantly changing and we need to train the model online.

6.2 Deep Q-Network

As mentioned in section 5.3, we train a model using week 1 trading data on AUDUSD. During training, we evaluate the model every 200 episodes using 20-hour data. We use google cloud to train the model for 2000 episodes, which takes about 10 hours in total. The running loss as well as training and evaluation rewards are as follows.

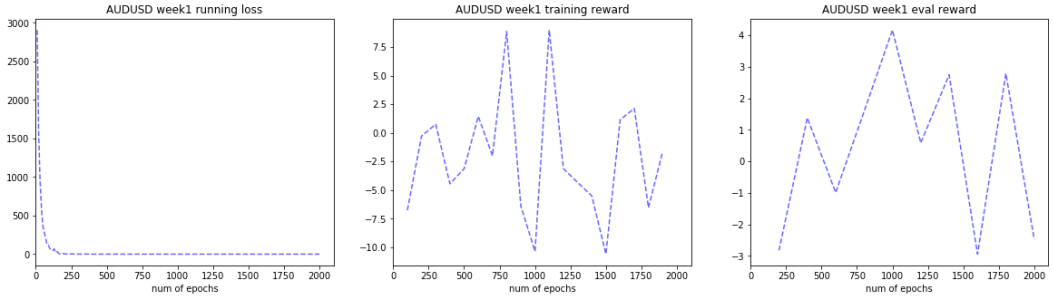


Figure 17: DQN performance on AUDUSD

As shown in figure 17, the running loss of DQN decreases monotonically while the training and evaluation rewards fail to improve accordingly. We use a 48-hour test dataset to further inspect the out-of-sample performance. On the test set, the agent decides to do nothing but keep the neutral position (0). We think the divergence between loss and reward is because the Q network may successfully model the long-term discounted returns of a state, while it's not as proper for 1-hour cumulative returns. Also, we constrain the agent to choose only from a finite action set, which may constrain the model's flexibility and prevent it from making active trades.

Given the result, we believe that the direct reinforcement learning is more promising as a FX trading strategy in practice.

7 Conclusion and next steps

Our experiments show that our direct reinforcement learning model can generate positive returns on certain currencies. Moving forward, we will briefly discuss the possibility of applying the model to a real-life trading environment. We will detail our thoughts on source of alpha, execution potential, as well as risk management settings.

7.1 Source of Alpha

Our model can be seen as a feed-forward neural network, which takes a set of features constructed by most recent FX prices and outputs a trading decision between -1 and 1. Similar to traditional FX trading strategies, we target to mine FX movement patterns and generate returns by identifying price momentum or reversion. We believe our current return is attributable to the following sources:

- Dataset: We use a high-frequency second-level trading dataset which may contain more signal compared to minute-level or daily data. In addition, we combine the bid/ask offerings from 5 liquidity providers, and feed the best prices into the reinforcement learning model. By narrowing down the bid-ask spread, we are more likely to generate profit.
- Trading duration: As a high-frequency trading strategy, our model stays in the market and trades consecutively for 1 hour. The risk due to extended trading duration may generate excess returns.
- Dynamic decision making algorithm: One key advantage of the reinforcement learning algorithm is that it can make decisions sequentially. Such dynamic planning capacity enables the model to optimize the total return over a desired period. By incorporating the transaction cost (bid-ask spread), the model may learn to balance between capturing signals and controlling transaction cost, which helps to generate positive returns over a period.

7.2 Execution Discussion and Risk Management

Figure 13 shows the profit and loss when executing the strategy and consistently trading on the EURUSD pair over 48 hours. Similar to the way we train the model, transactions are made on a per-second basis. The model will take in the features at each time step and output an executable position.

One important factor when executing trades is to limit negative risk. Therefore, the maximum downturn of the model should be restricted to a small number. It is possible to make the model risk-averse by restricting the amount of the transactions it can make. We incorporate this philosophy into the design of our reward. By incorporating the bid-ask spread, the model will learn to be more conservative since any type of position, long or short, is inherently risky because going to that position requires losing the spread.

Besides the well-defined reward function, we incorporate the previous position to the model, with a self-learned coefficient. We are hoping that, by taking into consideration of the last position, the model will learn to preserve the position and not to take aggressive movements more directly.

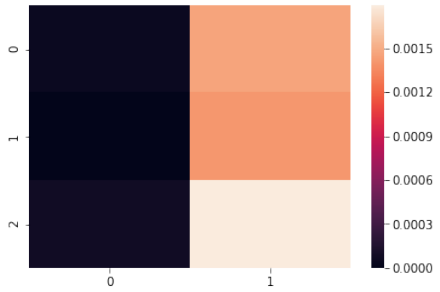


Figure 18: Comparing the relative importance of the previous position to other input features

As is seen in figure 18, the previous position (on the right) is much more important than the gradient in average(on the right). Also, the coefficient is positive. It can be concluded that the model learns to preserve its past position and tends not to take exaggerated movements.

However, the largest concern is that the offline model is only exposed to the data a week prior, and although the model could capture the market structure well, it is not sufficient to prevent the exposure to systematic risk. In imitation learning, the cascading error will occur if the agent is exposed to an environment it has never seen before. It is similar here for our strategy, when the market deviates beyond the expected behaviour, the model will be experiencing the cascading failure by capturing wrong signals. Human intervention is required when the market experiences some strong deviations from expected.

7.3 Retrospective Discussion

Looking back on our project, we have learned the following things:

1. Direct reinforcement learning can generate positive returns for FX trading.
   Through our experiments, direct reinforcement learning yields positive returns on all the three weeks test set for EURUSD and on week 2 test set for GBPUSD. The results are reasonable because our strategy is essentially a trend-following strategy and the features we use resembles the factors in traditional factor models.
2. Our Deep Q-network does not work well on FX trading.
   During our experiments, the agent of the DQN learns to take neutral positions only and break even on the test set. The reasons may lie in the limited flexibility due to the discrete action set, the indirect learning objective, and confusing features.
3. Deep models perform better.
   Before introducing DNNs, we first implemented our baseline direct reinforcement learning model with a single linear layer. It is not surprising to find that deep model structures work better than naive shallow ones.

If we could continue with the project, here are the potential next steps:

1. Incorporate better features
   One reason why DQN not works may lie in feature selection. Therefore, we could implement feature engineering such as time series analysis to discover the most important features. We can also incorporate other advanced features like technical features to make the agent act more like a professional trader.
2. Build a better architecture
   We are now using a 3-layer neural network as model structure, which can be further extended with residual blocks to build a deeper model, or with LSTM to learn with longer time lags.
3. Continue training and hyperparameter tuning
   Currently we are training with the data of 1 week. So we can continue training with data of a longer time span to increase the generalization ability of our model. Another thing we can do is to improve the model performance.
4. Add an online learning scheme
   We can also convert the training of our algorithms into an online learning fashion, which most resembles real-time trading, to reduce the workload of backtesting.

While FX trading is a difficult field to penetrate, we believe that reinforcement learning shows significant promise. We believe with more targeted knowledge such as important feature extraction and architectural modifications, there is significant promise in the use of reinforcement learning in FX trading and algorithmic trading as a whole–especially as markets become more and more technologically advanced and deep learning applications become more prevalent.

References

[1] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

[2] Enguerrand Horel, Virgile Mison, Tao Xiong, Kay Giesecke, and Lidia Mangu. Sensitivity based neural networks explanations. *arXiv preprint arXiv:1812.01029*, 2018.

[3] Chien Yi Huang. Financial trading as a game: A deep reinforcement learning approach. *arXiv preprint arXiv:1807.02787*, 2018.

[4] Matthew Johnson. Forex algorithmic trading: Understanding the basics. `https://www.investopedia.com/articles/forex/053115/basics-forex-algorithmic-trading.asp`, 2018.

[5] J. McGeever. Daily fx trading volume falls 5.5 pct to $5.1 trillion -bis. `https://uk.reuters.com/article/bis-currency-idUKL8N1BC4PL`, 2019.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[7] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4):875–889, 2001.

[8] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680. ACM, 2006.

[9] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

GitHub codebase: Reinforcement Learning for FX Trading