STACKED DEEP-Q NETWORKS FOR FOREX TRADING

James Harbour
Math and CS
University of Virginia
gtr8rh@virginia.edu

Deniz Olgun
Neuro and CS
University of Virginia
epv2kz@virginia.edu

ABSTRACT

This paper presents an innovative approach to automated trading in the Forex market, leveraging the power of Deep Q-Networks (DQNs) and sentiment analysis to inform trading decisions. Recognizing the Forex market's continuous operation and the abundance of public data it offers, we propose a multi-tiered machine learning strategy aimed at executing profitable trades with minimal exposure to underlying asset price volatility. Our methodology involves the development of three interconnected DQNs: the first processes price time series data to recommend actions (buy, hold, or sell); the second combines price data with sentiment analysis of global news articles to forecast currency pair price movements; and the third synthesizes outputs from the first two networks to finalize trading decisions. This strategy capitalizes on the predictive power of DQNs and the informative value of real-time news sentiment, aiming to achieve non-zero risk-adjusted returns that are uncorrelated with market movements. Preliminary results, drawn from backtesting and anticipated live market trials via the MetaTrader API, suggest a promising avenue for automated Forex trading strategies that enhance profitability while managing risk. This work builds on existing literature by incorporating reinforcement learning into Forex trading and extends previous applications of sentiment analysis in cryptocurrency markets to the realm of fiat currencies.

***K*eywords** Forex · Deep Q learning · Sentiment Analysis

1 Introduction

1.1 Problem Definition

There is considerable interest in developing automated trading strategies for financial instruments. Foreign currency exchanges (Forex) are one of the most liquid financial markets, trading continuously with high volume for 24 hours 5 days a week. In this setting where vast amounts of public data are available and information asymmetry is low, we anticipate that machine learning strategies would be useful.

Our input data will be a medium-frequency price time series on currency pairs against USD and global news articles. Our analysis will transform these data to produce a policy decision to buy, hold, or sell each currency pair at the given timepoint. The portfolio of currency pairs will then be hedged against a basket of options that minimizes exposure to the prices of the underlying assets.

2 Proposed Idea

For this application project, we plan to explore applications of Deep Q-networks (DQN) in various architectures. We also plan to use a text embedding and sentiment analysis library to classify news articles as an input feature. Our tentative plan is to develop two data pre-processing networks which will feed into a third policy network which will actually perform the trade using a pre-existing API.

3 Experiment Setup

Experiment Setup: describe datasets (forex data), preprocessing (normalization, RNN data / selecting 5 datapoints), hyper-parameters (just implement randomized search), optimization methods (IDEK), different model architectures with varying levels of complexity,

3.1 Datasets

Our experiment utilizes intraday foreign exchange time series data for EUR/USD collected from the Oanda broker via the python package *pyfinancialdata* [1]. We utilize frequencies of both hourly and minute-to-minute. The dataset consists of Open, High, Low, Close values over the time interval.

INCLUDE GRAPH OF FOREX

3.2 Preprocessing

For input into the LSTM, following the Box-Jenkins approach to time series modelling we analyzed the partial autocorrelation function for our minute-level time-series to determine the appropriate window size. From figure **??**, we chose to use a time lag of 5.

INCLUDE PACF FIGURE

We then transformed our time-series to a series of over-lapping subsequences. For example, if we used a time lag of 3, then we would perform the following transformation:

$$[1, 2, 3, 4, 5, 6, 7] \mapsto [1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6], [5, 6, 7]$$

As financial data has an underlying meaning associated to its magnitude, we felt any normalization to be unwise. As such, we left the time-series largely untouched when feeding into the deep-Q network.

4 Model Architectures

The intuition of feeding in a sequence of currency values into a model and predicting, say, the next few values naturally leads one to consider a recurrent neural network (RNN) architecture. These are a class of neural networks specifically designed to handle sequential dataand, unlike traditional feedforward networks, RNNs can capture dynamic temporal behaviors as they have an internal state that serves as a form of memory. In practice, one can treat an RNN as a feedforward neural network with a hidden layer block for each time step. This unravelling intuition can be seen in figure 1.

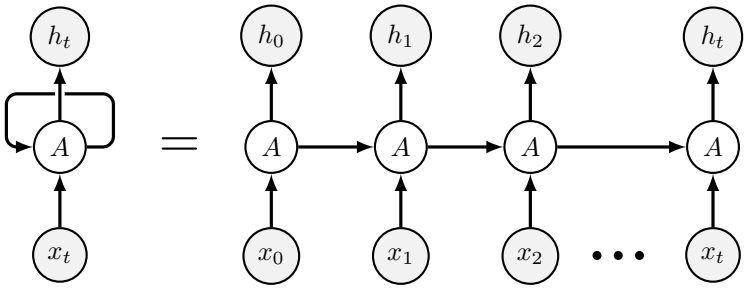


Figure 1: Unravelling of a simple recurrent neural network

Applying backpropogation to this type of time-unfolded feedforward network is known as backpropogation through time (BPTT). One main difficulty that arises from this model when running BPTT is that the multiplicative nature of chain rule leads to exponentially growing or decaying gradients over time, thus presenting a massive barrier to learning long-term dependencies. In working with foreign exchange data over multiple years, this initially somewhat minor problem becomes incredibly pertinent and thus forces us to consider more complicated recurrent architectures.

4.1 LSTM

Long Short-Term Memory (LSTM) networks were specifically designed to tackle this gradient decay/explosion by introducing memory cell and gating mechanisms to selectively remember and forget certain data. Intuitively, this selectiveness in memory prevents the content held within the entire network from growing too large as time moves. Mathematically, BPTT applied to this architecture leads to linear combinations of gradients instead of multiplicative combinations, thus explaining why LSTM is an ample drop-in fix over simple RNN. For the reader's edification, we have included a depiction of an LSTM cell below in figure 2

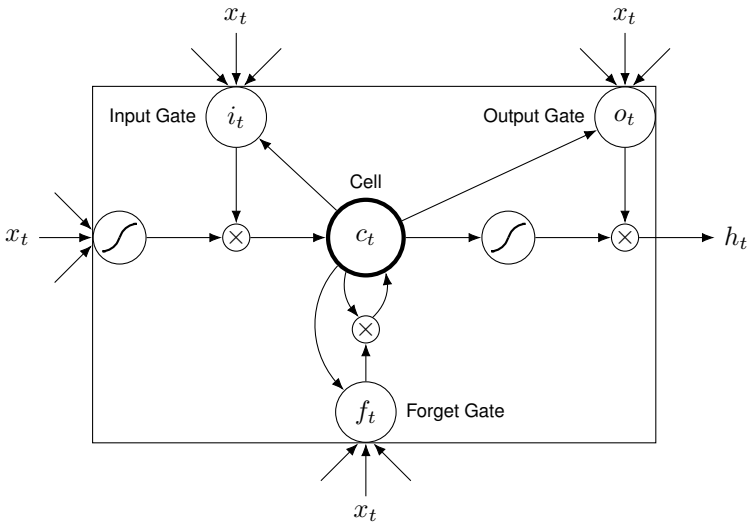


Figure 2: LSTM cell

4.2 Deep Q-Networks

One point that is not at all obvious from a time-series point of view is how to make optimal trading decisions based up predictive data. Even with a way to look ahead into the future, one requires the extra input of some type of trading strategy to actually convert this data-edge into genuine profit. This is difficult in practice and the main methods in practice, known as "technical analysis," are seen as financial astrology by many. As such, one way to sidestep this problem is to simply outsource the trading decision making to a learning algorithm. This places us in the setting of reinforcement learning, namely Q-learning.

Q-learning is a fundamental algorithm in reinforcement learning that enables an agent to learn the optimal action to take in a given state. The core concept revolves around the Q-function, denoted as $Q(s, a)$, which estimates the expected total reward an agent can obtain by starting from state s, taking action a, and following an optimal policy thereafter.

4.2.1 State and Action Spaces

To formalize the environment in which the agent operates, we define two key components:

- The **state space**, $\mathcal{S}$, encompasses all possible states the agent can encounter. It can be either discrete or continuous, depending on the nature of the problem.
- The **action space**, $\mathcal{A}$, represents the set of actions available to the agent at each state. Similar to the state space, the action space can be discrete or continuous.

4.2.2 Updating Q-Values: The Bellman Equation

The Q-values are iteratively updated using the Bellman equation, which forms the foundation of Q-learning:

$$Q_{\text{new}}(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{1}$$

where α represents the learning rate, controlling the step size of each update, and γ is the discount factor that balances the importance of immediate and future rewards. The equation adjusts the Q-value based on the received reward r_{t+1} and the maximum expected future reward from the subsequent state s_{t+1}.

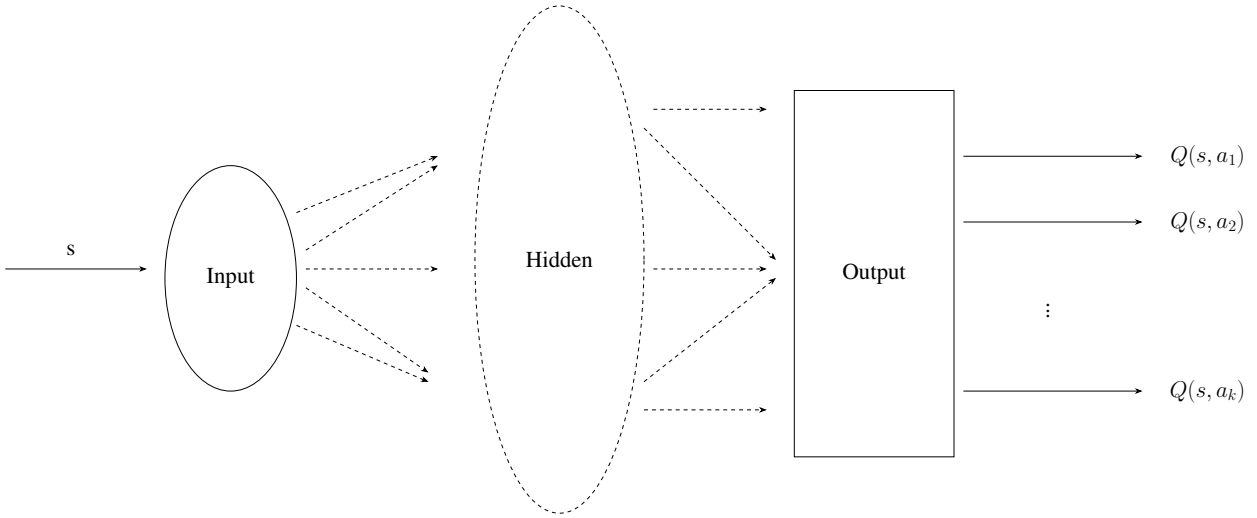
4.2.3 Integrating Deep Learning: Deep Q-Networks (DQNs)

While traditional Q-learning uses a tabular approach to store Q-values, Deep Q-Networks (DQNs) leverage the power of deep neural networks to approximate the Q-function. This extension enables Q-learning to handle high-dimensional state spaces and overcome the limitations of the tabular method.

4.2.4 DQN Architecture

A typical DQN architecture consists of the following components:

- **Input Layer:** Accepts the state representation of the environment as input.
- **Hidden Layers:** Comprise multiple layers that extract relevant features and learn complex representations.
- **Output Layer:** Produces the estimated Q-values for each action in the given state.



4.2.5 Training Procedure

The training process of a DQN involves the following key steps:

1. **Experience Replay:** The agent's experiences, represented as tuples $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$, are stored in a replay buffer. During training, mini-batches of experiences are randomly sampled from this buffer, promoting stability and reducing correlations between samples.
2. **Loss Calculation:** The temporal difference error is used to define the loss function:
$$L = \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)\right)^2 \tag{2}$$
where θ represents the parameters of the DQN, and θ^- denotes the parameters of a separate target network.
3. **Target Network Updates:** To stabilize the learning process, the target network parameters θ^- are periodically updated with the parameters of the main DQN θ. This helps mitigate the issue of moving targets and enhances the stability of the algorithm.

By integrating deep learning with Q-learning, DQNs have achieved remarkable success in various domains, enabling agents to learn complex policies directly from high-dimensional sensory inputs.

4.3 Hyperparameters and Optimization

5 Expected Outcome

We would like to finish implementing this algorithm about 3 weeks before the final due date. This way, we can go beyond backtesting on historical data and test the model's performance live on real markets. We will route the output of our terminal DQN through the MetaTrader API to execute our trades on either a paper account or real assets. We hope to capture non-zero risk-adjusted returns that are uncorrelated with the underlying market.

6 Related Works

- Our initial motivation for this project is the recent paper of Otabek and Choi [2] which combines deep Q-networks, and sentiment analysis to create a trading strategy for bitcoin. This algorithm improves upon previous systematic bitcoin algorithms mainly due to the sentiment analysis addition, and we suspect that such a strategy can be adapted to the world of fiat currencies.
- Ayitey Junior et. al. in [3] survey the current literature on the application of deep learning, both standard and reinforcement-based, to foreign exchange markets. From this analysis, we see that the vast majority of machine learning applications to this setting are via standard neural network architectures. Only a small minority (2 as of now) utilize reinforcement learning in any fashion.
- As the first application of RL to the Forex market, Carapuço, Neves, and Horta in [4] utilize a pure deep-Q based policy network and obtain quite admirable returns. This provides motivation for further refinement of the method as it performs somewhat well without inital added complexity.

7 Datasets

7.1 Financial Datasets

For access to the Forex market, we utilize the REST API inside of MetaApi for communicating with the exchange, obtaining both historical and real-time currency data, and executing trades

7.2 Sentiment Analysis Datasets

For data input into the predictive DQN (similar to [2]), we utilize relevant news articles acquired via publicly available new agency APIs (see this database)

8 Requirements for Final Report

- Experiment Setup: describe datasets (forex data), preprocessing (normalization, RNN data / selecting 5 datapoints), hyper-parameters (just implement randomized search), optimization methods (IDEK), different model architectures with varying levels of complexity,
- Experiment Results: LSTM (sharpe, backtesting graph), DQN (sharpe, backtesting graph), DQN + LSTM (sharpe, backtesting graph)
- Results Analysis: Do your results meet your expectations? If the results are expected and good, please identify the important factors that lead to success. Otherwise, please identify the unexpected issues in the proposed ideas
- Conclusion: Short bit about things we have learned: can say surprising just how much went into non-ML tasks (setting up environment, finding good data, cleaning it, fitting it into pipeline) than actually implementing and testing models. Though we expected that to take some time, it was roughly 90/10. We found python to be an unforgiving language for large projects and that the lack of compile-time type checking together with often cryptic error messages difficult. Setting up ML dashboards to compare model was hard, we were often confused about whether our implementations were actually using the GPU or a single thread, . We were tempted to make our own financial backtesting networks and subsequently found

References

[1] Max Williams. financial-data. https://github.com/FutureSharks/financial-data, 2020.

[2] Sattarov Otabek and Jaeyoung Choi. Multi-level deep q-networks for bitcoin trading strategies. *Scientific Reports*, 14(1):771, Jan 2024.

[3] Michael Ayitey Junior, Peter Appiahene, Obed Appiah, and Christopher Ninfaakang Bombie. Forex market forecasting using machine learning: Systematic literature review and meta-analysis. *Journal of Big Data*, 10(1):9, Jan 2023.

[4] João Carapuço, Rui Neves, and Nuno Horta. Reinforcement learning applied to forex trading. *Applied Soft Computing*, 73:783–794, 2018.