Cryptocurrency Automated Trading Agent

Project blueprint

Author: Roberto Lentini

2021

Table of Contents

**Introduction2**

Reinforcement Learning2

Reinforcement Learning in Finance3

**Algorithms4**

Categories of Reinforcement Learning Algorithms5

Value-Based Algorithms5

Deep Q-Learning (DQN)5

Double DQN (DDQN)6

Dueling DQN7

Gated DQN (GDQN)7

Rainbow DQN (RDQN)7

Value-Based Algorithms in Automated Trading8

Policy Gradient Algorithms8

Advantage Actor Critic (A2C)8

Gated Deterministic Policy Gradient (GDPG)9

Proximal Policy Optimization (A2C)10

Policy Gradient Algorithms in Automated Trading11

Data List11

References13

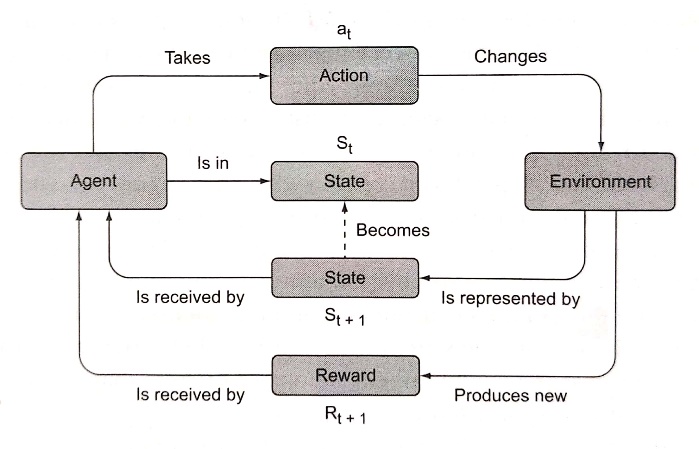
**Reinforcement Learning**

Machine learning is a sub field of artificial intelligence which uses past data to be able to create models that can think in an intelligent manner. The architectures of a standard machine learning model, such as a supervised model, will work in a linear fashion as a static function. If we use a supervised learning model to solve an image classification problem, we will be passing data into the model to get the output of the image category. It is essentially passing values to a model to get an output.

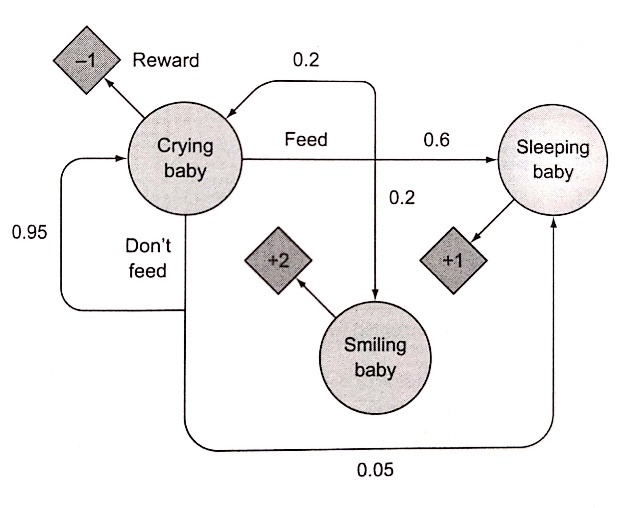
Reinforcement learning is a sub field of machine learning that seeks to solve a different kind of problem that standard machine learning can not solve efficiently. It tackles control tasks. A control task is a scenario where an algorithm needs to learn how to act and not just classify or predict like in standard machine learning. All the data in a control task lives in time and space. The algorithm’s future decisions will be influenced by its passed decision. This means that a reinforcement learning algorithm framework is built in a loop format as opposed to linear format.

Let’s evaluate how both types of algorithms would handle a control task to understand the difference between machine learning and reinforcement learning. To be able to make a self driving car we would need to teach the car how to drive properly in a simulation. If we were to tackle this issue using a supervised learning method, we would need people to label the data frame by frame. This would allow the algorithm to learn the correct decisions for each scenario the car would face. This would accumulate to endless hours of labeling, and it would be virtually impossible to label each scenario the car would face. Instead, if we use reinforcement learning we do not need to tell the car what the right thing to do at every timepoint is. The cars “brain” would learn how to drive and act in different scenarios through repetition. From its previous decisions the cars “brain” will learn how to act for future scenarios.

The essence of reinforcement learning can be described as such: we have the learning algorithm which takes an action while attempting a control task, i.e. driving a car. The learning algorithm will get a reward which will positively or negatively influence the action that it took. This is much like trying to teach a dog a new trick. A person will give the dog a treat when it successfully completes the trick which will help reinforce the trick in the dog’s brain.

Understanding the basics of the reinforcement learning architecture requires knowing a handful of terms, these consist of state, action, reward, environment, and agent. The agent is the part of a reinforcement learning algorithm that processes inputs to decide what action to take. In our self driving car example the car’s “brain” would be what we refer to as the agent. The environment is the world which the agent lives in. Essentially the environment is what generates the input data for our agent. In the self driving car example, the environment refers to the simulated world the car (our agent) is situated in. The state is information from the environment at a certain timestamp which the agent uses to decide how to act. In the self driving car example the state could just be the open road at one point or if the car encounters a stop sign then the state would be updated, and the agent will need to decide what to do based on this input from the environment. If we think of a game of chess the state of the board would be the position of all the chess pieces at a certain time stamp. An action is the decision taken by the agent which will change its environment and lead to a new state. If our self driving car sees a stop sign and stops then it has taken the action to stop. A reward is a positive or negative signal that is given to the agent after it has taken an action. If our self driving car took the action to stop when it evaluated the current state, seeing a stop sign, then it would receive a higher reward as opposed to not having stopped (under the assumption that we are teaching the agent how to drive obeying to traffic laws). The agent’s goal is to maximize its reward therefore it’s good actions will be reinforced using a higher reward.

**Figure 1.0 A diagram of a reinforcement learning framework from [1] p.25 . An agent takes an action in its current state. This action will modify the environment and bring a new state to the action. The action will also cause a reward which the agent will use for its next action decision. These rewards are used to reinforce the agents learning.**

We can now build from the general architecture of reinforcement learning and look at the theoretical processes that allow it to work. The theory behind reinforcement learning builds on the Markov process, also known as the Markov chain. The Markov process is a system which describes a sequence of possible events and states. This sequence of all possible states in an environment is known as the state space. The states in the system need to fulfill the Markov property which means that the system does not have a memory, a future state can only depend on the current systems state. The Markov Decision Process (MDP) is an extension to the Markov process where the system now takes into account an action space, set of all possible actions, and rewards. Let’s look at an example of a Markov Decision Process. Let’s say we have a baby with the following state space: crying, sleeping and smiling. The action space will consist of feed and don’t feed. We will also attach rewards to the states. If the baby is smiling the reward will be +2, if the baby is sleeping the reward will be +1 and if the baby is crying the reward will be -1. The entire system can be seen in the following diagram.

**Figure 2.0 A diagram of the Markov Decision Process (MDP) [1] p.49.**

If we have a crying baby and decide not to feed it then there’s a 0.95 probability that the baby will remain in the current state of crying giving the system a reward of -1. If we feed a crying baby, then there is a probability of 0.6 that the baby will transition to a state of sleeping giving the system a reward of +1. The rest of the system follows the same pattern. The goal of our agent in a reinforcement learning problem is to maximize the reward of the Markov Decision Process and this is the way the agent learns how to act in a control task.

The reinforcement learning agent uses value functions and a policy function to be able to maximize its reward in a Markov Decision Process. The policy function is a mapping from states to actions given those states. The value functions refer to the expected reward of being in some state or taking some action. We have two value functions: the state-value which maps state to the expected reward and the action-value function, also know as the Q function, which maps a pair of an action and a state to an expected reward.

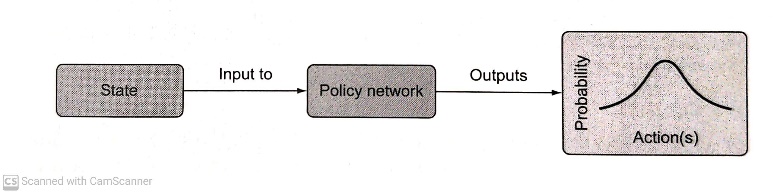
The policy is a strategy the agent uses to determine what action to perform given a certain state. Given a certain state, the agent will follow a policy to make a choice of which action to take. An example of a policy would be the epsilon greedy policy. An agent using the epsilon greedy policy will refer to it’s list of action-state pairs given by the action-value function and with probability the agent will take a random action and with probability 1 – will take the action given a state that maximizes its reward. The balance between taking random actions and taking the action with the highest reward is adjusted to find a balance between exploration and exploitation. We want our agent to sometimes explore the environment to find new action-state pairs, but we also want it to learn to exploit the environment to take the best actions to lead to the highest rewards.

Prior to getting into the different reinforcement learning algorithms and their mathematics, I will define an episode and deep reinforcement learning. An episode consists of a sequence of states reaching to the terminal state, the state that ends the game. An episode in a game of chess would be when a check mate occurs. It is important to note that some environments are episodic in nature, a game of chess, and others are not because they have no clear terminal state, the stock market. A deep reinforcement learning algorithm is reinforcement learning that uses a neural network as a policy function. The reason neural networks are of high interest as a policy function for reinforcement learning is because a neural network is great for recognizing high level features in data. This makes it useful for when the state space is too large to be completely known.

**Reinforcement Learning in Finance**

The basic knowledge regarding the general architecture of deep reinforcement learning gained in the last section will allow us to see how it fits in the financial world. Let’s associate the financial terms to the deep reinforcement learning terms we learned in the previous section. Our investor is our agent, the investors’ goal is to maximize his or her profits, the reward. To do so the investor needs to decide the appropriate time to buy, hold and sell a certain asset, this is our action space, based on the current market indicators (ie. price), the state space. The advantage of a deep reinforcement learning agent is that it can automate the transactions for the investor as well as watch many different data sources at the same time. The agent will be able to know what the state space is at all times, whereas an investor might be busy doing something else such as eating dinner. The hope is that the agent will be able to find patterns in the market that will allow it to take advantage of the cryptocurrencies volatility by buying dips and selling when the market is overly bullish, the agent is deemed successful if it can beat the buy and hold strategy which is my current strategy. It is important to take into consideration the transaction fees the agent will incur and remove that from the total reward. In the following section we will evaluate different algorithms that will be used to teach our agent how to act in the cryptocurrency market.

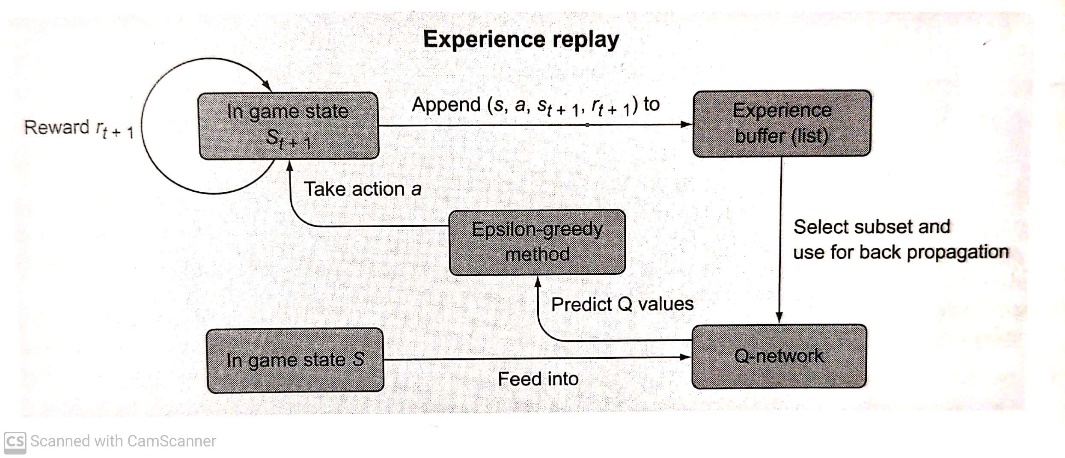
**Categories of Reinforcement Learning Algorithms**

The two categories of reinforcement learning which will be evaluated for this project are value-based algorithms and policy gradient algorithms. A value-based algorithm in deep reinforcement learning is an algorithm that approximates the action-value function using a neural network, the Q-network. This Q-network will output approximations, given state, of which action has the highest expected reward then a policy will be used to choose which action to take (i.e. the epsilon-greedy policy). We will be looking at the following value-based algorithms: Deep Q-Learning (DQN), Double DQN (DDQN), Dueling DQN. Gated DQN (GDQN) and Rainbow DQN (RDQN). Policy gradient algorithms in deep reinforcement learning train a neural network to output an action directly. Policy-gradient algorithms will take a state and will return a probability distribution over the possible actions. We will be looking at the following policy gradient algorithms: Advantage Actor Critic (A2C), Proximal Policy Optimization (PPO), Gated Deterministic Policy Gradient (GDPG).

**Figure 3.0 Shows the difference between a policy gradient algorithm (top diagram) and a value-based algorithm (bottom diagram). Diagrams are taken from [1] p.90-91.**

**Value-Based Algorithms**

**Deep Q Learning (DQN)**

Deep Q-Learning is the simple case of value-based algorithms when it comes to deep reinforcement learning. It was introduced to the world in a paper published by DeepMind [3]. We will use it as a base model to compare to the other reinforcement learning models and see the performance difference. The general idea of deep Q-learning is to feed an action and a state to the Q-network which will output a predicted Q-value. The Q-value is the expect reward for the given action-state pair. The action is then taken and the target Q-value as well as the predicted Q-value are used to calculate the loss function (i.e. Mean Squared Error). The Q-network is then updated using a gradient descent method (i.e. Stochastic Gradient Descent). The DQN algorithm is prone to a problem called catastrophic forgetting which is when we train our Q-network with small batches of data at a time, this causes the new data to be forgotten and thus our agent will not truly be learning to act in the stock market it will just be basing its decision off of the last ones and forgetting everything it previously learned. To solve the issue of catastrophic forgetting we will add experience replay to our algorithm. Experience replay is a mechanism which will allow our agent to remember what it had previously learned through batch training and thus reducing the noise of our algorithm. To further increase the stability of the DQN algorithm we will introduce a target Q-network. The target Q-network produces Q-values which are used to train, through backpropagation, the main Q-network. The target Q-network parameters are never trained but they are instead periodically synchronized with the main Q-networks parameters. This lag between the Q-network stops the agent from being overly influenced by one move.

**Figure 4.0 A diagram of DQN algorithm using an experience replay component and an epsilon-greedy policy [1] p.82.**

Algorithm [2] p.138:

1. Initialize parameters for the Q-network,, and target Q-network, , with random weights and an empty replay buffer (the list where we are going to store past experiences and use a random subset of these to update the Q-network as opposed to just using the most recent experience to update the network).
2. Use a policy to choose which action to take (i.e., epsilon greedy, with probability ε choose a random action, , otherwise or the softmax policy).
3. Perform action, , in the environment and observe the next state, .
4. Store the experience in the replay buffer.
5. Sample a mini batch of experiences from the replay buffer.
6. For every experience in the buffer, calculate the target if the agent reached the terminal state this step, else . The discount factor,, determines how much the agent will weigh future rewards as opposed to the more immediate rewards.
7. Calculate loss notice that this loss function uses the computed target network Q-value from step 6.
8. Update Q-network using a gradient descent algorithm (in our case stochastic gradient descent (SGD)) by minimizing the loss in respect to parameter models.
9. Every N-steps copy weights from main Q-network to the target Q-network.
10. Repeat from step 2 until convergence.

**Double Deep Q-Learning (DDQN)**

DeepMind introduced the DDQN as an improvement to the existing DQN algorithm [4]. A DQN will tend to overestimate Q-values due to the part of the target equation, . The paper by DeepMind [4] proposed the solution to change the target Q-values to the following formula . They prove in their paper that this fixes the issue of overestimation in DQN. Everything in the DDQN algorithm remains the same as DQN aside from the mentioned formula.

**Dueling Deep Q-Learning (Dueling DQN)**

Another issue with the DQN algorithm which DDQN does not solve is that when an agent experiences a bad state which leads to a low reward instead of lowering the , state-value function which maps state to the expected reward, the agent will store the low reward in its memory due to some action by updating the , the action-value function. The issue with this process is that the agent will keep returning to this unpromising state till all the actions leading to that state are updated in the Q function [5]. This is computationally inefficient. To solve this issue and increase the algorithms convergence speed the Dueling DQN was introduced [6].

The DQN network will take the features from the Q-network and outputs a vector of Q-values for each action and then a policy will choose an action to take [2] p.217 . The Dueling DQN will take features from the Q-network and separate them into two different streams. One stream will be predicting the , state-value numbers, and the other will predict the advantage values. The advantage is a computation calculating how much more of a reward an action from a certain state will bring us, . To obtain the Q-value we add the value to the and subtract the mean value of the advantage given use the following formula . The rest of the algorithm is the same as the DQN.

**Gated Deep Q-Learning (GDQN)**

The Gated DQN algorithm consists of the same algorithm as the DQN but as opposed to using a Convolution Neural Network (CNN) it uses Gated Recurrent Unit (GRU). The GRU networks are a type of Recurrent Neural Network (RNN). An RNN is used when context is important. They have sequential memory. The issue with basic RNN’s is that they are victims of the vanishing gradient problem, as the network is trained it will be influenced much less by prior events. The GRU network solves this issue by learning to keep relevant information in the model’s memory using different gates. We could also have used Long/Short Term Memory networks (LTSM) but these tend to be longer to train than the GRU networks.

**Rainbow Deep Q-Learning (RDQN)**

The Rainbow Deep Reinforcement Learning (RDQN) is an ensemble algorithm which combines many of the algorithms we discussed and some more we have not gone over. The RDQN was introduced to try to get the most out of all the improvements to the DQN algorithm [7]. It unites the following algorithms DQN, DDQN, Dueling DQN, Noisy DQN [8], Prioritized Experienced Replay [9], Multi-Step DQN [10], Categorical DQN [11]. This algorithm is computationally costly but performs very well compared to any of the algorithms discussed individually on Atari games.

Algorithm [5]:

Hyperparameters: — batch size, , ,— parameters of support,— target network update frequency, — multi-step size, — degree of prioritized experience replay, — importance sampling bias correction for prioritized experience replay, — neural network, SGD optimizer.

support grid

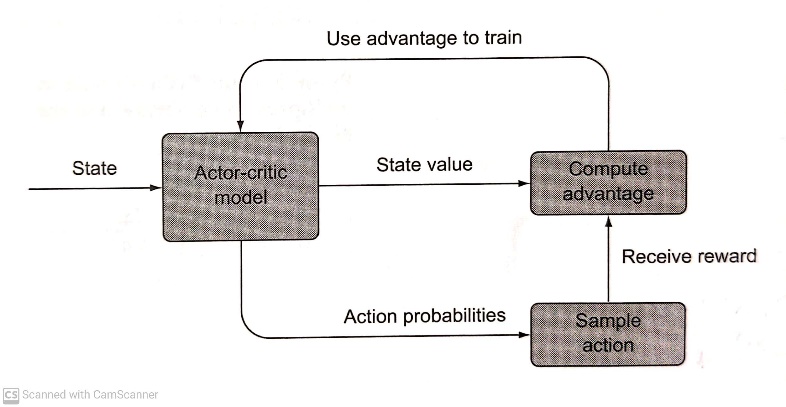
1. Select input
2. Observe experience
3. Construct N-step experience and add it to the experience replay with priority .
4. Sample batch from experience replay using probabilities
5. Compute weights for the batch, where is the size of the experience replay memory
6. For each experience from the batch compute the target
7. Project target onto support
8. Update experience priorities
9. Compute loss:
10. Make a gradient descent step
11. If

**Value Based Algorithms in Automated Trading**

The DQN algorithm has been extensively studied in automated trading and it is now often used as a comparison algorithm to test the performance of its improved counterparts [12] [13] [14]. The basic DQN algorithm has been shown to outperform trading strategies such as the buy and hold, decision tree-based method [14] as well as the double cross over strategy [15]. Since these earlier studies, the DQN algorithm has been improved in various ways bringing to life new algorithms such as DDQN and Dueling DQN. Both the DDQN and the Dueling DQN outperform the DQN agent in automated trading [13]. Researchers decided to use Gated Recurrent Unit neural network with the DQN strategy (GDQN) and it outperformed the Turtle trading strategy [16]. Not much has been researched in using the Rainbow DQN (RDQN) as an automated trading agent but the one paper published [17] about it showed that it outperformed the DDQN, DQN as well as the Dueling DQN. Although, it suffered issues in highly volatile market conditions which means that it will need quite a bit of adjustments for the cryptocurrency market. It will need to be fed more high quality data to help it trade in volatile conditions. RDQN greatly outperforms many other value-based algorithms in Atari games [7].

**Policy Gradient Algorithms**

**Advantage Actor Critic (A2C)**

The advantage actor critic method is a combination of both policy gradient algorithms and value-based algorithms. The advantage actor critic algorithm has a neural network which returns two vectors, one for the policy and one for the value, . The policy function, the actor, and the value function, the critic, are independent. The policy function is going to return a probability distribution over the possible actions the agent can take in the given state. The value function will output the expected return for the agent given a state and an action. Essentially, the actor will take an action and the critic will inform the actor on the performance of the action. Using the state-value and the action probabilities, the advantage will be calculated, . The advantage value is then used to train the model.

**Figure 5.0 A diagram of the Advantage Actor-Critic algorithm [1] p.125.**

The N-step learning is an addition to the A2C algorithm. All it means is that the loss is calculated, and the parameters are updated at every N step. With N-step learning instead of updating the neural network at the end of full episodes we update it during the episodes too. This will cause the target value for the critic network to be more accurate and less biased.

Algorithm [18]:

1. Acquire a batch of transitions using the current policy (either a finite episode or a truncated one).
2. For each state encountered, compute the discounted sum of the next rewards and use the critic to estimate the value of the state encountered steps later .
3. Update the actor
4. Update the critic to minimize the TD error between the estimated value of a state and its true value.
5. Repeat until convergence

**Gated Deterministic Policy Gradient (GDPG)**

Deterministic Policy Gradient (DPG) was introduced to tackle control tasks in the continuous environment [19]. It is a type of actor critic method that is deterministic as opposed to stochastic like the A2C. In the A2C algorithm the policy network (actor) outputs the probability of an action given a state . The A2C algorithm also computes the actor’s expected gradient using this formula .The DGP policy network outputs a single action since it is now deterministic, . With this action the actor gradient can be calculated using the following formula . As opposed to the A2C algorithm the DPG only integrates relative to the state since the action is deterministic. The Gated Deterministic Policy Gradient (GDPG) is a DPG which uses Gated Recurrent Units for its policy and value neural networks.

**Proximal Policy Optimization (PPO)**

The PPO alternates between sampling data by acting within the environment and by optimizing the objective function by using SGD [20]. The PPO is another type of actor-critic method with a few adjustments. These modifications include a change in the method used to estimate policy gradients and a new way of estimating the advantage. As opposed to taking the log probability of the action as we would in the A2C model, we use the ratio between the new and the old policy scaled by the advantages, . The advantage function is now updated using the following formula:   
.

Algorithm [5]:

**Hyperparameters**: — batch size, — rollout size, **n\_epochs** — number of epochs, **ε** — clipping parameter, — critic neural network, — actor neural network, **α** — critic loss scaling, SGD optimizer.

1. Obtain a roll-out of size using policy , storing action probabilities as .
2. For each experience from the roll-out compute advantage estimation:
3. Perform n\_epochs passes through roll-out using batches of size B; for each batch:
   1. Compute critic target
   2. Compute critic loss
   3. Compute critic gradients
   4. Compute importance sampling weights
   5. Compute clipped importance sampling weights
   6. Compute actor gradient
   7. Make a step of gradient descent using.
4. Till convergence

**Policy Gradient Algorithms in Automated Trading**

Policy gradients method are more effective at learning the dynamics of trading than Q-learning, a value-based method [21]. The GDPG algorithm was more stable than the GDQN algorithm for stock trading [17]. Not much research has been done on the PPO algorithm and its application to stock trading. The PPO algorithm has been shown to outperform other online policy gradient methods playing Atari games, therefore it will be worth considering for the trading agent. An ensemble algorithm using the PPO, A2C, DDPG was used to create a trading agent and it outperformed the Dow Jones Industrial Average index (DJIA) and the Min-Variance portfolio allocation strategy. While the ensemble model outperforms the PPO algorithm, the PPO algorithm performs very promisingly [22].

**Data List**

The research published on cryptocurrency trading just use cryptocurrency prices, sentiment analysis and some blockchain information mostly separately. A trading algorithm using all the different data sources has not yet been evaluated. A big advantage with cryptocurrency is the number of API’s which provide a lot of great data sources that can describe the blockchain very well. The following data list was created based on research evaluating the main drivers of cryptocurrency prices as well as what other researchers used for their trading agent. I have also added some data which could be useful to the agent. It is important to note that the data sources may need to be changed because of the evolving landscape of Ethereum. Since Ethereum is moving to a proof of stake concept as its protocol is updated mining difficulty will become irrelevant since mining will be terminated. A correlation analysis will be done to evaluate which data sources will remain in the model to avoid any collinearity issues which we might run into. The list also contains a link to the location of the data sources.

* **Market Capitalization**: the total dollar value of a company’s outstanding shares. (share priced x total outstanding shares) source: [23][24] site: <https://etherscan.io/chart/marketcap>
* **Daily Transactions**: the number of transactions on the ether network. source: [25] [24] site: <https://etherscan.io/chart/tx>
* **ERC20 Daily token transfer**: ERC20 is a standard used for creating and issuing smart contracts on the Ethereum blockchain. sub tokens. source: none site: <https://etherscan.io/chart/tokenerc-20txns>
* **Unique Addresses** : total distinct number of address on the ether network. source: [24] site: <https://etherscan.io/chart/address>
* **Trading Volume**: number of coins exchanged in a day. Source: [23] [24] site: <https://coinmarketcap.com/currencies/ethereum/historical-data/>
* **Google Trends:** the amount of searches for the word “Ethereum”. source: [25] [26] analysis site: <https://trends.google.com/trends/explore?geo=CA&q=ethereum>
* **Market Beta:** this will calculate the volatility of Ethereum source: [27] site: this will need to be calculated manually.
* **Average Transaction Fee:** the daily average amount in USD spent per transaction on the Ethereum network. Source: [27] [24] site: <https://etherscan.io/chart/avg-txfee-usd>
* **Network Difficulty:** the mining difficulty of the Ethereum network. source: [27] [24] site: <https://etherscan.io/chart/difficulty>
* **Hash Rate:** the measure of the processing power of the Ethereum network. source: [27] [24] site: <https://etherscan.io/chart/hashrate>
* **Bitcoin Price:** the price of bitcoin source: none site: <https://coinmarketcap.com/currencies/bitcoin/>
* **Order Book Data:** spread, ask/bid depth, depth difference, ask/bid volume, volume difference, weighted spread, ask/bid slope. source: [28] site: I would need to collect this data over a period of time
* **Tweet Volumes:** the amount of tweets that use the word Ethereum. source: [26] site: <https://bitinfocharts.com/comparison/tweets-btc-eth-ltc-doge.html#3y>
* **News Sentiment:** the general news sentiment of a crypto news aggregator regarding the crypto market. source: [29] site: <https://cryptonews-api.com/>
* **Average Block Size:** average block size in bytes of the Ethereum blockchain. source: [24] site: <https://etherscan.io/chart/blocksize>
* **Network Transaction Fee Chart:** number of ether paid as a fee for the Ethereum network. source: [24] site: <https://etherscan.io/chart/transactionfee>
* **Bitcoin Hash Rate:** the hash rate of bitcoin. source: none site: <https://www.blockchain.com/charts/hash-rate>
* **Macro Economics:** S&P500, Eurostoxx, DOW30, NASDAQ, Crude oil, SSE, Gold, VIX, Nikkei225, FTSE100. Source: [24] site: unknown
* **Daily Active Ethereum Address:** unique addresses that were on the Ethereum network as senders or receivers source: none site: <https://etherscan.io/chart/active-address>

**References**

[1] Brown, Brandon, and Alexander Zai. *Deep Reinforcement Learning in Action*. Shelter Island, NY: Manning Publications Company, 2020.

[2] Lapan, Maxim. *Deep Reinforcement Learning Hands-on: Apply Modern RL Methods to Practical Problems of Chatbots, Robotics, Discrete Optimization, Web Automation, and More*. Birmingham: Packt, 2020.

[3] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wiestra, and Martin Riedmiller. “Playing Atari with Deep Reinforcement Learning.” *DeepMind Technologies*, December 19, 2013. https://arxiv.org/abs/1312.5602v1.

[4] van Hasselt, Hado, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-Learning.” *Thirtieth AAAI Conference on Artificial Intelligence* 30, no. 1 (March 2, 2016). https://doi.org/https://ojs.aaai.org/index.php/AAAI/article/view/10295.

[5] Ivanov, Sergey, and Alexander D'yakonov. “Modern Deep Reinforcement Learning Algorithms,” July 6, 2019. https://doi.org/https://arxiv.org/abs/1906.10025.

[6] Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. “Dueling Network Architectures for Deep Reinforcement Learning.” *Proceedings of The 33rd International Conference on Machine Learning* 48 (June 20, 2016): 1995–2003. https://doi.org/https://proceedings.mlr.press/v48/wangf16.html.

[7] Hessel, Matteo, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. “Rainbow: Combining Improvements in Deep Reinforcement Learning.” *Thirty-Second AAAI Conference on Artificial Intelligence*, April 2018. https://doi.org/https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/viewPaper/17204.

[8] Fortunato, Meire, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, et al. “Noisy Networks for Exploration,” July 9, 2019. https://doi.org/https://arxiv.org/abs/1706.10295.

[9] Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver. “Prioritized Experience Replay,” February 25, 2016. https://doi.org/https://arxiv.org/abs/1511.05952.

[10] Sutton, Richard S. “Learning to Predict by the Methods of Temporal Differences.” *Machine Learning* 3, no. 1 (1988): 9–44. https://doi.org/10.1007/bf00115009.

[11] Bellemare, Marc G, Will Dabney, and Remi Munos. “A Distributional Perspective on Reinforcement Learning.” *Proceedings of the 34th International Conference on Machine Learning* 70 (August 6, 2017): 449–58. https://doi.org/https://proceedings.mlr.press/v70/bellemare17a.html.

[12] Lucarelli, Giorgio, and Matteo Borrotti. “A Deep Reinforcement Learning Approach for Automated Cryptocurrency Trading.” *IFIP Advances in Information and Communication Technology*, 2019, 247–58. https://doi.org/10.1007/978-3-030-19823-7\_20.

[13] Li, Yuming, Pin Ni, and Victor Chang. “Application of Deep Reinforcement Learning in Stock Trading Strategies and Stock Forecasting.” *Computing* 102, no. 6 (2019): 1305–22. https://doi.org/10.1007/s00607-019-00773-w.

[14] Chakole, Jagdish Bhagwan, Mugdha S. Kolhe, Grishma D. Mahapurush, Anushka Yadav, and Manish P. Kurhekar. “A Q-Learning Agent for Automated Trading in Equity Stock Markets.” *Expert Systems with Applications* 163 (2021): 113761. https://doi.org/10.1016/j.eswa.2020.113761.

[15] Sattarov, Otabek, Azamjon Muminov, Cheol Won Lee, Hyun Kyu Kang, Ryumduck Oh, Junho Ahn, Hyung Jun Oh, and Heung Seok Jeon. “Recommending Cryptocurrency Trading Points with Deep Reinforcement Learning Approach.” *Applied Sciences* 10, no. 4 (2020): 1506. https://doi.org/10.3390/app10041506.

[16] Wu, Xing, Haolei Chen, Jianjia Wang, Luigi Troiano, Vincenzo Loia, and Hamido Fujita. “Adaptive Stock Trading Strategies with Deep Reinforcement Learning Methods.” *Information Sciences* 538 (2020): 142–58. https://doi.org/10.1016/j.ins.2020.05.066.

[17] Wu, Xing, Haolei Chen, Jianjia Wang, Luigi Troiano, Vincenzo Loia, and Hamido Fujita. “Adaptive Stock Trading Strategies with Deep Reinforcement Learning Methods.” *Information Sciences* 538 (2020): 142–58. https://doi.org/10.1016/j.ins.2020.05.066.

[18] Vitay, Julien. Deep reinforcement learning. Accessed October 5, 2021. https://julien-vitay.net/deeprl/ActorCritic.html.

[19] Lillicrap, Timothy P, Jonathan J Hunt∗, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. “Continuous Control with Deep Reinforcement Learning.” *conference paper at ICLR 2016*, July 5, 2016. https://doi.org/https://arxiv.org/abs/1509.02971.

[20] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal Policy Optimization Algorithms,” August 28, 2017. https://doi.org/https://arxiv.org/abs/1707.06347.

[21] Sadighian, Jonathan. “Deep Reinforcement Learning in Cryptocurrency Market Making,” November 21, 2019. https://doi.org/https://arxiv.org/abs/1911.08647.

[22] Yang, Hongyang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. “Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy.” *SSRN Electronic Journal*, 2020. https://doi.org/10.2139/ssrn.3690996.

[23] Alessandretti, Laura, Abeer ElBahrawy, Luca Maria Aiello, and Andrea Baronchelli. “Anticipating Cryptocurrency Prices Using Machine Learning.” *Complexity* 2018 (2018): 1–16. https://doi.org/10.1155/2018/8983590.

[24] Jang, Huisu, and Jaewook Lee. “An Empirical Study on Modeling and Prediction of Bitcoin Prices with Bayesian Neural Networks Based on Blockchain Information.” *IEEE Access* 6 (2018): 5427–37. https://doi.org/10.1109/access.2017.2779181.

[25] Kristoufek, Ladislav. “What Are the Main Drivers of the Bitcoin Price? Evidence from Wavelet Coherence Analysis.” *PLOS ONE* 10, no. 4 (2015). https://doi.org/10.1371/journal.pone.0123923.

[26] Abraham, Jethin, Daniel Higdon, John Nelson, and Juan Ibarra. “Cryptocurrency Price Prediction Using Tweet Volumes and Sentiment Analysis.” *SMU Data Science Review* 1, no. 3 (2018). https://doi.org/https://scholar.smu.edu/datasciencereview/vol1/iss3/1/.

[27] Sovbetov, Yhlas. “Factors Influencing Cryptocurrency Prices: Evidence from Bitcoin, Ethereum, Dash, Litcoin, and Monero.” *Journal of Economics and Financial Analysis* 2, no. 2 (2018): 1–27. https://doi.org/https://papers.ssrn.com/sol3/papers.cfm?abstract\_id=3125347.

[28] Guo, Tian, Albert Bifet, and Nino Antulov-Fantulin. “Bitcoin Volatility Forecasting with a Glimpse into Buy and Sell Orders.” *2018 IEEE International Conference on Data Mining (ICDM)*, 2018. https://doi.org/10.1109/icdm.2018.00123.

[29] Vo, Anh-Dung. “Sentiment Analysis of News for Effective Cryptocurrency Price Prediction.” *International Journal of Knowledge Engineering* 5, no. 2 (2019): 47–52. https://doi.org/10.18178/ijke.2019.5.2.116.