The Picolo project

Adi Kancherla

November 26, 2017

version 1 (private release)

Whitepaper for the project can be found at [1]

Abstract

This is a work in progress. Methods and equations presented here will most likely change as the platform evolves. Current thinking towards methods for training AI algorithms that make price predictions for cryptos and take bets from users are discussed. Note that work presented here is not original research but merely an innovative way of using existing work of various authors. Appropriate attributions are made through out the paper and a full list of references can be found at the end. AI algorithms used here work on two distinct types of data: unstructured in the form of text and structured in the form of exchange data and bets on the platform. While the research on sentiment analysis on unstructured data is vast and several off the shelf solutions are available, we explore an approach based on LSTM networks. For structured data, a novel approach is proposed where the state of the platform (all current bets, buzzing topics, randomly timestamped snapshots of past platform state) is reduced to a "game state" and presented as a Markov Decision Process. Incentive decay functions that govern the payouts made to users for contributing on the platform are discussed.

1 Introduction

Blockchain technology is taking the world by storm and is increasingly capturing people's attention. The total market cap of all the cryptocurrencies and tokens was about $4 billion at the beginning of 2017 and has grown close to $300 billion as of November 2017. It is hard to recall such a rate of growth for any industry in history. Much like a startup company that found viral success, the crypto space has much to deal with the unexpected growth in order to take on traditional asset classes and challenge the status quo. While there are excellent infrastructural projects in the works like plasma [2] for solving scaling issues and filecoin [3] for solving storage issues, there are few projects that take on the problems experienced by the drivers of this boom: investors. In the accompanying whitepaper [1] we discussed the problem in greater detail and the solution we are building. This paper details some of the initial thoughts on the implementation and underlying math for the features presented in the whitepaper. The incentive payout system is partly inspired by concepts from

Mechanism design theory [4] while AI algorithms draw upon research in the fields of reinforcement learning techniques [5] and artificial neural networks [6].

2 AI algorithms

Some factors that affect prices of cryptos are listed below. These factors are by no means exhaustive but provide a framework within which mechanisms to analyze them can be discussed. See the sub sections where some techniques are presented. Factors affecting the prices of crypto assets:

- Is this a base crypto like bitcoin or ethereum? Base cryptos are used to buy other cryptos. This increases demand for it until the seller of the other crypto decides to sell the base crypto.
- Hype cycle. Current news or sentiment for a crypto. New developments like signing a partnership or launching a new product that increases its utility.
- Profit taking. When an asset gains a lot of value from its previous "base", profit taking can take place.
- Loss cutting.
- Regulatory concerns. Cryptos are a new development and there is regulatory uncertainty surrounding them. Developments in this regard can hugely affect prices.
- Increased awareness. As more and more people start paying attention to the space, more money flows in which can affect volatility.
- Technological advancement. While blockchain technology has a huge potential to transform various industries, there remain concerns regarding scalability, privacy, security, ease of use etc. Progress in addressing these concerns will have a lasting impact on asset prices.

Following methods can be used to effectively analyze the impact of above factors on crypto prices.

2.1 Analyzing unstructured data

Unstructured data consists of the following types of content:

- Posts, comments, questions etc.
- Memes, gifs, videos etc.
- Content from third parties in the form of tweets etc.

While there is considerable amount of research in multimodal sentiment analysis a.k.a extracting sentiments from videos and GIFs, those methods are not explored in the current iteration of this paper. See [7] and [8] for examples. Here a method to perform sentiment analysis on textual data using an LSTM (Long Short Term Memory) is described. LSTM is a type of a recurrent neural

net, that can learn dependencies in an arbitrarily long sequence of events. RNNs (Recurrent neural network - Fig. 1 [9]) retain a state that can represent information from an arbitrarily long context window. Further, they can simultaneously model sequential and time dependencies on multiple scales. This success is attributed to their ability to learn hierarchical representations. But RNNs suffer from vanishing and exploding gradients problem (see [10]) making them hard to train. To solve this, an LSTM uses a memory cell (Fig. 2). A memory cell has a node with a self-referencing edge of fixed weight one, preventing the gradient from exploding or vanishing across time steps. See Fig. 3 for an RNN with two memory cells.

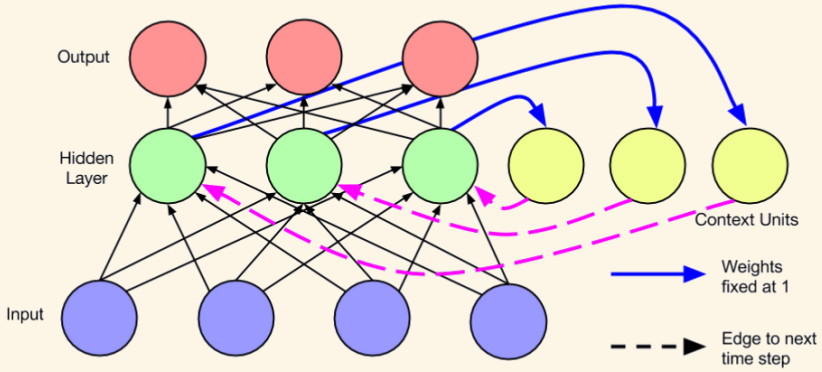

Figure 1: A recurrent neural network

2.1.1 Methodology

Price predictions from sentiments is done in three distinct steps:

- Individual sentiment analysis on a unit of data (a single post, comment etc.)
- Combining the vector output from the above step with additional data (e.g price at the time of sentiment inference) and adding it to a sequence
- Analyzing the sequence when it reaches a desired length, by feeding it to an LSTM that outputs predictions

Length of the sequence can be varied to get predictions for different time scales. For e.g, a shorter sequence can be used to predict prices in the short term where as a longer sequence can be used to predict prices in the long term.

2.1.2 Unit data analysis

There exists a large body of research on sentiment analysis of tweets and other social media data. See [11] and [12] for two of the most cited works. Most

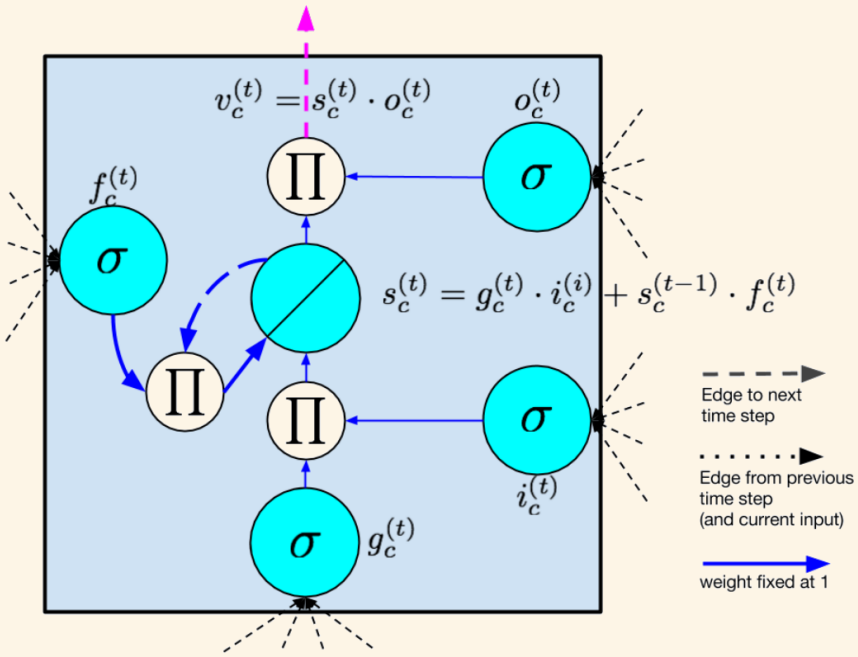

Figure 2: LSTM memory cell

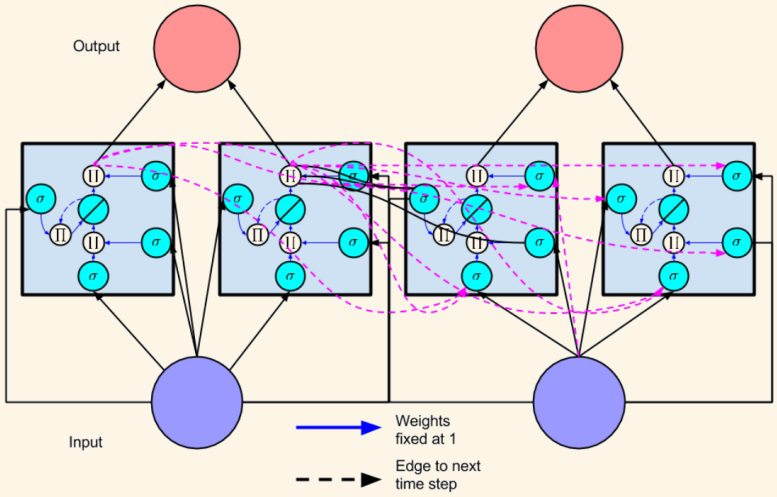

Figure 3: A recurrent neural network with two memory cells expanded across two time steps

of the research including these two works use SVM (Support Vector Machine) classifiers to train and run a model. Pre trained models are readily available for

common tasks like inferring sentiment from movie reviews which can be reused with a few modifications. However, we propose using a simple cloud API in this paper. One such API is Google's natural language API. It takes in a piece of text and outputs the sentiment expressed as a score and magnitude. Score represents the overall sentiment and magnitude measures how strong the sentiment is. Score ranges from -1.0 (negative) to 1.0 (positive) where as magnitude ranges from 0.0 to ∞. Higher the magnitude, higher the strength of the sentiment. For example, a score of 0.5 and a magnitude of 6.2 indicates a strong mildly positive sentiment. Output from this task is represented as a two dimensional vector of the form [s, m] and is added to a sequence.

2.1.3 Sequence analysis

Once we have a desired length sequence, an LSTM is employed to perform analysis and output predictions. A simplified view of an LSTM cell is shown in Fig. 4 [13]. It has the following components:

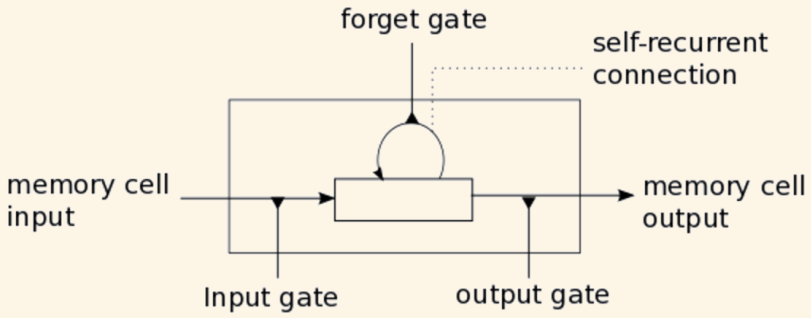

Figure 4: Simplified view of an LSTM memory cell

- Input node i. Takes input vector x_t of the form [s, m] from the current time step and output of the hidden layer h_{t-1} from the previous time step
- Input gate g. Similar to input node, takes x_t and h_{t-1}. Its value multiplies the value of the input node i
- State s. State of the memory cell
- Forget gate f. Controls how much of a previous state has to be remembered and used in the current calculation
- Output gate o. Outputs the final value of computation in the current time step

These equations fully determine the computation [9]:

$$
\begin{aligned}
g_t &= \phi(W^{gx}x_t + W^{gh}h_{t-1} + b_g) \\
i_t &= \sigma(W^{ix}x_t + W^{ih}h_{t-1} + b_i) \\
f_t &= \sigma(W^{fx}x_t + W^{fh}h_{t-1} + b_f) \\
o_t &= \sigma(W^{ox}x_t + W^{oh}h_{t-1} + b_o) \\
s_t &= g_t \odot i_t + s_{t-1} \odot f_t \\
h_t &= \phi(s_t) \odot o_t
\end{aligned}
$$

where

W s are weight matrices between components

b s are biases

ϕ is the tanh function $\frac{e^x - e^{-x}}{e^x + e^{-x}}$

σ is the sigmoid function $\frac{1}{1+e^{-x}}$ and

$\odot$ is point multiplication

Activations from the memory cell layer are fed to the output layer of the RNN which produces final predictions. The output layer employs a softmax function that calculates the probabilities of a set number of price ranges. If a_i are the activations, then softmax $S(a_i)$ is given by

$$
S(a_i) = \frac{e^{a_i}}{\sum_{i=1}^{i} e^{a_i}}
$$

2.1.4 Training

The LSTM can be trained using various sequence lengths α to predict prices after various time scales γ and β [14] (see Fig. 5). For example, to train the LSTM to predict prices 2 days into the future, we might consider a sequence length of 10000 where the latest sentiment vector in the sequence is generated a day prior to the prediction date . So we have $\alpha = 10000$, $\beta = 48$ and $\gamma = 24$ (time is measured in hours). A large number of such sequences and corresponding prices are taken from the past and are used for training.

2.2 Analyzing structured data

Structured data consists of:

- Price feed from exchanges
- Quantified signal from sentiment analysis
- Order book from exchanges
- Current bets
- Snapshots from the past that serve as histories

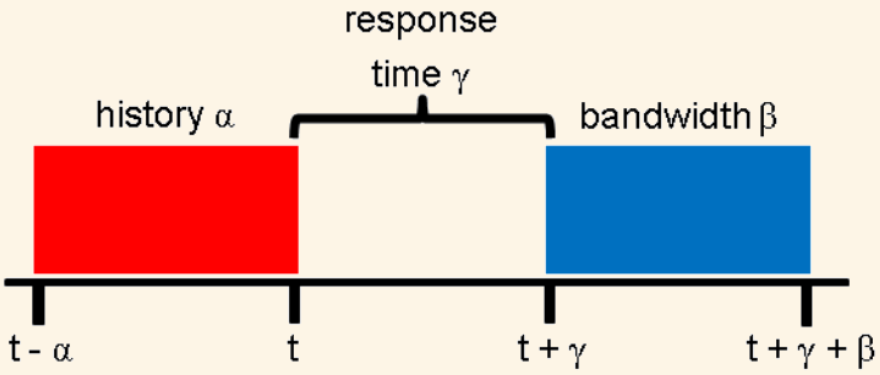

Figure 5: Training for predictions on different time scales

An AI agent has access to all this information and its objective is to predict the future state of the system. The agent must be able to learn which of its actions are desirable based on rewards that can take place arbitrarily far in the future [15]. Here, reward can simply be the accuracy of its predictions. The agent is trained to maximize the reward. Problems with delayed reinforcement are well modeled as Markov decision processes (MDPs) [16]. An MDP is characterized by a set of states, a set of actions, a state transition function and a reward function. A value function is defined by

$$V(s) = \max_a \left(R(s,a) + \gamma \sum\nolimits_{s' \in S} T(s,a,s') V(s') \right), \forall s \in S$$

where

S is the set of states

A is the set of actions

R is the reward function $R : S \times A \to \Re$

T is the state transition function $T : S \times A \to \pi(S)$ where π is the probability distribution function over the set of states

γ is the discount parameter

$T(s,a,s')$ is the probability of state change from s to s' when action a is taken

In the real world, state cannot be completely observed. Even the partially observable state is so large and action space infinite that they cannot be represented efficiently by any data structure. Hence a function approximator is constructed from training data and is used to predict future states and recommend actions. An artificial neural network serves as an excellent approximator due to its general structure and availability of various training algorithms. Thus the architecture of the system consists of three components: First, a state representation $s = f(s)$ that encodes raw input s (snapshots from the past, live

observations from exchange data, current bets etc where f is a neural network) into an internal (abstract, hidden) state S. Second, a model M with state transition function $T(s, a, s')$, reward function R with rewards r and internal discounts γ. Third, a value function V that outputs values $v(s)$ representing the future from internal state s onwards. The system is applied by unrolling the model M multiple planning steps to produce internal rewards, discounts and values Fig. 6 [17]. Here g is the overall output:

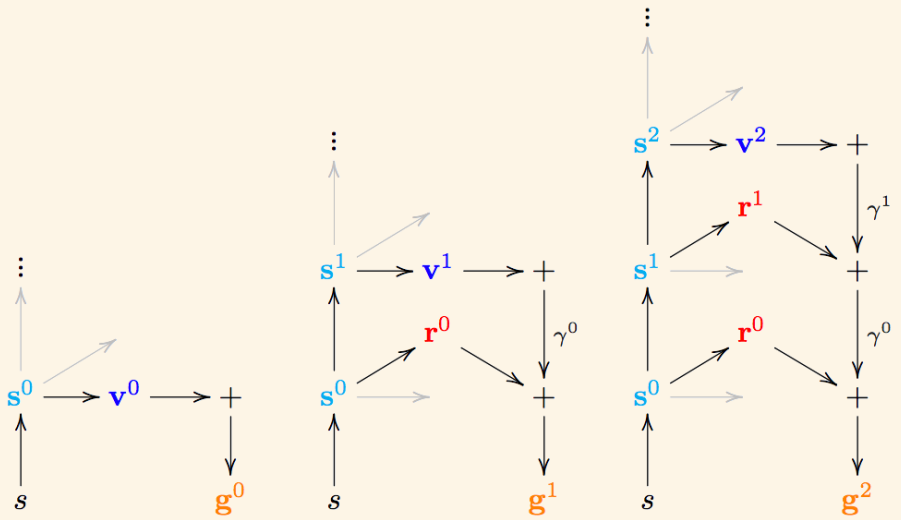

Figure 6: Model rollouts

$$g_n = r_1 + \gamma_1(r_2 + \gamma_2(...(r_{n-1} + \gamma_{n-1}(r_n + \gamma_n v_n))...))$$

Initially all parameters θ of the system are set to random values or a prior domain knowledge. Then updates are made via stochastic gradient descent by minimizing the loss function,

$$L_n = \frac{1}{2}\left(V_E(g \mid s) - V_M(g_n \mid s)\right)^2$$

where

V_E is the set of values observed in a real environment and

V_M is the set of values predicted by the model M

2.2.1 Monte-Carlo Tree Search

Since the state space is large, updates above cannot be efficiently implemented in the same way as in perfect information games. In cases like these, Monte-Carlo simulation provides an effective mechanism both for tree search and for state updates, breaking the curse of history and allowing much greater scalability [18]. An extended UCT [19] algorithm provides a computationally efficient best-first search that focuses its samples in the most promising regions of the search space.

If prior domain knowledge is available, the algorithm narrowly focuses the search on promising states without altering asymptotic convergence. The algorithm uses a simulator G as a generative model of the POMDP. The simulator provides a sample of a successor state, observation and reward, given a state and action,

$$G(s_t, a_t) = (s_{t+1}, o_{t+1}, r_{t+1})$$

and is used to generate sequences of states, observations and rewards. These simulations are then used to update the value function V. The extended UCT algorithm in [18] uses a search tree of histories instead of states (Fig. 7). A history is a sequence of actions and observations, $h_t = \{a_1, o_1, ..., a_t, o_t\}$ available to the AI agent. The agent's action-selection behavior can be described by a policy, $\pi(h, a)$, that maps a history h to a probability distribution over actions, $\pi(h, a) = P(a_{t+1} = a \mid h_t = h)$. The history tree contains a node $T(h) = \langle N(h), V(h) \rangle$ for each represented history h. $N(h)$ counts the number of times that history h has been visited. $V(h)$ is the value of history h, estimated by the mean return of all simulations starting with h. New nodes are initialized to $\langle V_{init}(h), N_{init}(h) \rangle$ if domain knowledge is available, and to $\langle 0, 0 \rangle$ otherwise. During the simulation actions are selected to maximize

$$V'(ha) = V(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}}$$

Here c is a constant that determines the trade off between exploration and exploitation. $c = 0$ corresponds to greedy exploitation. We approximate the future state for history h_t from K sample states, $s_i \in S_t, 1 \leq i \leq K$ by,

$$F(s, h_t) = \frac{1}{K} \sum_{i=1}^{K} \delta_{ss_i}$$

where δ_{ss_i} is the kronecker delta function. At the start of the algorithm, K samples are taken from an initial state distribution I_s(could be random). After a real action a_t is executed, and a real observation o_t is observed, the samples are updated by Monte-Carlo simulation. A state s is sampled from the future state $F(s, h_t)$, by selecting a state at random from S_t. This state is then passed into the generator G, to give a successor state s' and observation o'. If the sample observation matches the real observation, the new state s' is added to S_{t+1}. This process repeats until K states have been added to F. This approximation to the future state approaches the true future state with sufficient samples.

3 Incentive decay functions

Incentives are based on different categories of contribution. Two example categories are bringing new users to the platform and contributing original content. The incentive amount paid out for a user is dependent on how beneficial the user's contribution is to the platform relative to other users' contributions in that category. It is calculated by equation (1)

$$i = (w/W) * (I_c) \tag{1}$$

where

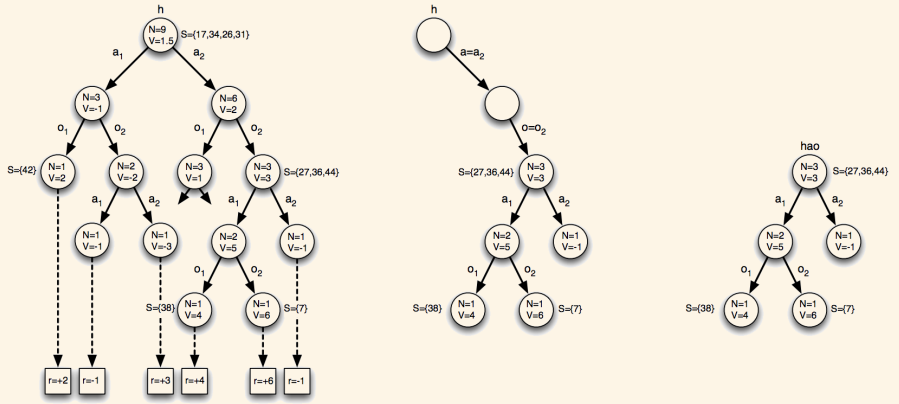

Figure 7: History tree search

i is the incentive amount paid to the user

w is the user's contribution

W is the sum of all contributions on the platform

I_c is the current incentive constant for category c

For example, in user growth category, W could be the total number of new users who joined the platform during time period t and w could be the number of new users brought to the platform by a user in the same time period. Incentive constant for a category can be calculated based on heuristics or a time/value based function. A possible heuristic for the user growth category is that the incentive constant halves for every 50,000 new users and can be calculated by the exponential decay function (2)

$$p_{k+1} = (1/2) * p_k \tag{2}$$

or reformulated as a log linear function (3)

$$\log(p_{k+1}) = \log(p_k) - 1 \tag{3}$$

where

p_{k+1} is the incentive amount at time $k+1$

p_k is the incentive amount at time k

and the number of users at time $k+1$ is 50,000 more than the users at time k

See Fig. 8 for a visualization of how the incentives decrease over time.

4 Conclusion

Initial implementations of AI algorithms that analyze structured and unstructured data are discussed. Unstructured data is analyzed in two steps: first, at a

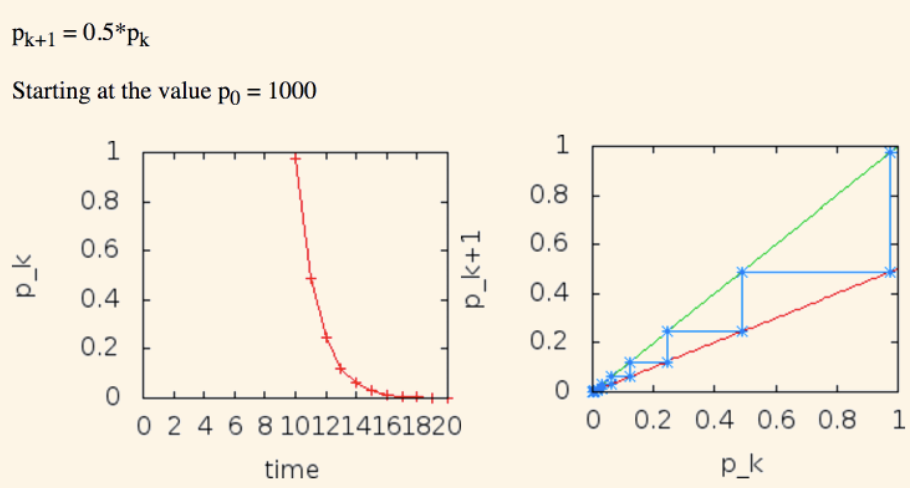

Figure 8: Incentive decay

unit level and second as a sequence by feeding it to an LSTM. Structured data consisting of live feed from exchanges, current and past bets on the platform amongst others is represented as a game state where an independent decision making agent learns to take actions that maximize its game score. A method of determining payouts to platform users is discussed where they are determined by the magnitude as well as the category of contribution.

References

[1] Adi Kancherla. Picolo: peer to peer and peer to AI betting for crypto tokens and currencies. https://picolo.ai/Whitepaper.pdf.

[2] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. https://plasma.io/plasma.pdf.

[3] Protocol Labs. Filecoin: A Decentralized Storage Network. https://filecoin.io/filecoin.pdf.

[4] The Royal Swedish Academy of Sciences. Mechanism design theory. https://www.nobelprize.org/nobel_prizes/economic-sciences/laureates/2007/advanced-economicsciences2007.pdf.

[5] Yuxi Li. Deep reinforcement learning: an overview. https://arxiv.org/pdf/1701.07274.pdf.

[6] Christos Stergiou and Dimitrios Siganos. Neural networks. https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html.

[7] Cai Zheng, Cao Donglin, and Ji Rongrong. Video (gif) sentiment analysis using large scale auto generated mid-level ontology. https://arxiv.org/pdf/1506.00765.pdf.

[8] Soujanya Poria, Erik Cambria, and Alexander Gelbukh. Deep convolutional neural network textual features and multiple kernel learning for utterance-level multimodal sentiment analysis. http://www.aclweb.org/anthology/D15-1303.

[9] Zachary C. Lipton and John Berkowitz. A critical review of recurrent neural networks for sequence learning. https://arxiv.org/pdf/1506.00019.pdf.

[10] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. https://arxiv.org/pdf/1211.5063.pdf.

[11] Saif M Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. https://arxiv.org/pdf/1308.6242.pdf.

[12] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. Sentiment analysis of twitter data. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.348.4054&rep=rep1&type=pdf.

[13] Pierre Luc Carrier and Kyunghyun Cho. Lstm networks for sentiment analysis. http://deeplearning.net/tutorial/lstm.html.

[14] Wei Peng, Ying Zhang, William Chan, Pang Wu, and Le T. Nguyen. Predicting collective sentiment dynamics from time-series social media. http://wan.poly.edu/KDD2012/forms/workshop/WISDOM2012/camera_ready/a6-nguyen.pdf.

[15] Michael L Littman, Leslie Pack Kaelbling, and Andrew W. Moore. Reinforcement learning: A survey. https://www.jair.org/media/301/live-301-1562-jair.pdf.

[16] Martijn van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. http://www.ai.rug.nl/~mwiering/Intro_RLBOOK.pdf.

[17] David et.al Silver. The predictron: End-to-end learning and planning. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/predictron.pdf.

[18] David Silver and Joel Veness. Monte-carlo planning in large pomdps. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Publications_files/pomcp.pdf.

[19] Levente Kocsis and Csaba Szepesvari. Bandit based monte-carlo planning. http://ggp.stanford.edu/readings/uct.pdf.