STAT 497 - Final Project

Concordia University

Bakr Abbas, Matthew Liu, Frederic Siino

Day Month Year

0 To do list

- Add footnotes
- Add links (to Jiang et al. and Sutton and Barto)
- Add references
- Add function description for policy gradient
- Add conclusion
- Determine if extra analysis should be added
- Add table of contents
- Review report

1 Introduction

One of the main challenges of managing a portfolio or a fund, is the decision process of how to allocate investment resources to each assets. The goal is to maximize returns but also to diversify, putting all the fund in the most profitable asset could be beneficial in the short run, but the possibility of that one asset crashing in value is larger than the possibility of multiple assets crashing in value. So diversifying is important to hedge against that outcome, and portfolio management in the modern day involves more automation than ever. Reinforcement & Deep learning methods are being used to optimize this decision making process.

Cryptocurrencies are decentralized digital assets that behave as an alternative to government issued money. They are being introduced into most investment portfolios, this paper address the challenge of managing five different cryptocurrencies with Bitcoin(BTC) being cash.Meaning that the four other currencies are quantified as the price relative to BTC. DIfferent methods will be used to manage the weights assigned to each asset in the portfolio. The first three methods will be based on the contextual bandits model.The final method will be based on a policy gradient model. A more realistic model would also take transaction costs into account, but this will be set aside for simplicity.

A previous research paper (Jiang, Xu, Liang, 2017) has been published on this exact topic based on a framework consisting of Ensemble of Identical Independent Evaluators (EIIE), a Portfolio-Vector Memory (PVM), an Online Stochastic Batch Learning (OSBL) scheme and a fully exploiting strategy. The actions were determined using a policy gradient method and the tested evaluators were a Convolutional Neural Network (CNN), a Recurrent Neural Network (RNN) and a Long Short-Term Memory Network (LSTM). Experiments were run with a trading period of 30 minutes and the RL framework was observed to behave far better than other optimization frameworks in the test period of 2017 but inferior to some frameworks in the test period of 2018.

In this project, we opt for a much simpler approach, relying instead on an action preference function and a policy gradient when framing the task as a contextual bandits problem. We do however borrow some aspects of Jiang, Xu and Liang such as definitions for states and returns.

2 Problem description

In this project, the environment is solely dictated by the data of the historic prices of all cryptocurrencies relative to BTC. In the experiments, a trading period of $T = 120$ minutes is used, and we only consider the closing prices of each period (which are equivalent to the opening prices of the next period). In addition, the experiments rely on two key assumptions (Jiang, Xu, Liang, 2017):

- Instant liquidity: The liquidity of the market allows each trade to be carried out immediately at the price listed when the order is placed.
- No market impact: The capital invested by the agent is so insignificant as to have no impact on the market.

This means that the sequence of states and transition probabilities are already predefined based on the data at hand. In particular, we define a price relative vector y_t of the tth trading period like Jiang, Xu and Liang, namely:

$$y_t := v_t \oslash v_{t-1} = \left(1, \frac{v_{2,t}}{v_{2,t-1}}, \ldots, \frac{v_{m-1,t}}{v_{m-1,t-1}}, \frac{v_{m,t}}{v_{m,t-1}}\right)^T \tag{1}$$

where $v_{t,i}$ denotes the closing price of the ith asset in the tth trading period. Note here that asset 1 and thus $v_{t,1}$ is always equal to 1. This is because the relative prices $v_{t,i}$ are relative to the first asset (in our case, BTC), also known as the cash. This price relative vector can then be used to calculate the reward, defined in general as

$$r_t = y_t \cdot w_{t-1} \tag{2}$$

where w_{t-1} is the portfolio weight vector at time $t-1$. This reward is effectively the change in portfolio value for period t. The action a_t at time t is the portfolio weight vector w_t, where the ith element is the proportion of asset i in the portfolio.

$$a_t := w_t \tag{3}$$

The goal of the agent is then to maximize the growth rate (or cumulative return) r_t of the portfolio over the experimental time frame.

In addition, we also define a state vector $S_T := (s_{1,T}, ..., s_{m,T})$ where $s_{i,T}$ denotes the state of the previous price changes of asset i at time T. Namely,

$$s_{i,T} = \sum_{t=T-n}^{t=T} \gamma^t y_{i,t} \tag{4}$$

where $y_{i,t}$ is the ith element of the previously defined price relative vector y_t, γ is a discount factor and n is the size of the history we consider. As such the state is a vector of n-step backward relative price changes for each asset discounted by a rate gamma.

Thus, based on the environment and problem definition, we have

$$p(s_{t+1}|s_t, a) := Pr[S_{t+1} = s_{t+1}|S_t = s_t, A_t = a] = 1 \; \forall a \in A \tag{5}$$

for the deterministic sequence of states

$$S_1, S_1, ..., S_{T-n}$$

which follows since the sequence $y_1, y_2, ..., y_T$ is determined by the data.

For the sake of simplicity, an additional assumption that there are no transaction fees is added. Thus the focus turns to wether or not the agent is able to learn the optimal re-weighting of the portfolio given the state at each time, i.e. the agent must identify promising assets. The question of transaction fees and practicality will be judged in a seperate analysis.

Lastly, since we let the agent run through all the available data when conducting an experiment, the problem becomes an episodic task where $t = 1, ..., T-n$ and T is the number of trading periods in our data set. Note here that t ends at $t = T - n$ since the window of the first n price changes needs to be considered as the starting state S_1, therefore reducing the number of states from T to $T - n$. In practice however, the problem is a continuing task since the agent is tasked to a manage a portfolio forever.

3 Implementations and Analyses

3.1 Method 1

3.1.1 Implementation

Method 1 is an ad-hoc method based on the contextual bandits problem. The portfolio allocation optimization problem is reframed as a k-armed bandits problem where we have one arm for each asset in the portfolio (including cash). The action preference method is used, where

$$H_t(a),\ a = 1, ..., k$$

are the action preferences for all k assets. However, a slight modification is done when defining actions and updating the preference functions. When computing probabilities of chosing a given action, we define $\pi_t(a|S_t)$ as

$$\pi_t(a|S_t) := Pr[A_t = a|S_t, R_1, ..., R_{t-1}] = \frac{e^{H_t(a)S_{a,t}}}{\sum_{b=1}^{k} e^{H_t(b)S_{b,t}}},\ a = 1, ..., k \tag{6}$$

where the previously defined state vector S_t from equation (4) is included due to the contextual nature of the problem. The n-size history of discounted price changes represented by S_t will thus change the probabilities of each asset based on whether or not their prices are observed to have declined or increased in the past n periods (uptrends increase preference, and downtrends decrease preference). Following the standard contextual bandits framework, this leads to the action at time t of

$$A_t = \arg\max_a \pi_t(a|S_t) \tag{7}$$

In the form of a portfolio weight vector, $A_t = w_t$ is a k-length vector where the A_tth co-ordinate is 1 and all other co-ordinates are 0. This means that rewards are

$$R_{t+1} := y_{t+1} \cdot A_t = y_{t+1,A_t} \tag{8}$$

i.e., the relative price change of asset A_t. The problem then reduces to a simple contextual bandits problem where one and only one arm is pulled by the agent at each time steps.

However, in the ad-hoc method the actual action taken by the agent at time t is the vector of contextual preference values, i.e.

$$A'_t = (\pi_t(1|S_t), \pi_t(2|S_t), ..., \pi_t(k|S_t))^T \tag{9}$$

Since all $\pi_t(k|S_t)$'s sum to 1, this is a valid portfolio weight vector, and we can set $A'_t = w_t$. The rewards are then computed as

$$R'_{t+1} := y_{t+1} \cdot A'_t \tag{10}$$

and the preference functions are updated based on R'_t as follows:

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R'_t - \bar{R}_t)(1 - \pi_t(A_t)) \tag{11}$$

$$H_{t+1}(a) = H_t(a) - \alpha(R'_t - \bar{R}_t)\pi_t(a),\ a \neq A_t \tag{12}$$

where $\pi_t(a)$ represents the non-contextual policy

$$\pi_t(a) := Pr[A_t = a|R_1, ..., R_{t-1}] = \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}},\ a = 1, ..., k \tag{13}$$

Intuitively, this method relies on the simplfying assumption that rewards produced by actions A'_t is a good enough approximation of rewards produced by actions A_t. In other words,

$$E[R'_t|S_t] \approx E[R_t|S_t]$$

In reality however, we can predict that this is far from the case (hence the ad-hoc nature of this method). It is safe to assume that computing the returns of a preference vector of assets is not the same as computing the total one-step return of the single most preferred asset. Nevertheless, the ad-hoc method provides a straightforward way to implement an algorithm learning to optimize a portfolio of assets since the softmax produces valid actions that represent the preference of the agent at time t based on S_t. However, the actual rewards used to update the preference function is only a weak approximation of the actual reward R_t defined in equation (8).

The function used to simulate a single episode of method 1 is `simulate_contextual1` and the parameters are as follows:

- `n_curren`: The number of currencies in the problem including cash (5 for this project)
- `n_steps`: The number of timesteps T in the episode. To use all available data set `n_steps = nrow(data)`
- `alpha`: Learning rate α when updating action preference function (as defined above)
- `window_size`: Size of historic prices to consider when computing S_t (`window_size` corresponds to the value of n)
- `discount`: Discount factor of previous relative price changes when computing S_t (`discount` corresponds to the value of γ)
- `data`: data that is used to run experiments

3.1.2 Analysis

Using this ad-hoc method, the agent consistently beats the market returns even when the learning rate and the window size. The reason behind this is due to one of the currencies, XRP, having a very steep increase in value (add footnote to refer to xrp spike). This increase causes a shift in the agent's preference to heavily to invest more in XRP during this spike. The action preferences for method 1 have a higher variance than method 2, which is noticable visually when looking at the graph of the preferences of each currency.

A grid search was conducted to find an optimal window size and alpha combination that would result in a higher agent return (add footnote to results). Combinations from alpha values ranging from 0.1 to 0.4, and window sizes ranging from 6 to 12. It was found that the best combination in this range was an alpha of 0.4 & a window size of 12, as the agent experiences a return of about 137% while the market return was roughly 85%. The larger alpha will cause the agent to prefer XRP even more during it's spike, which is between time step 2000 & 4000. This method does accomplish our main goal, which is to beat the market.

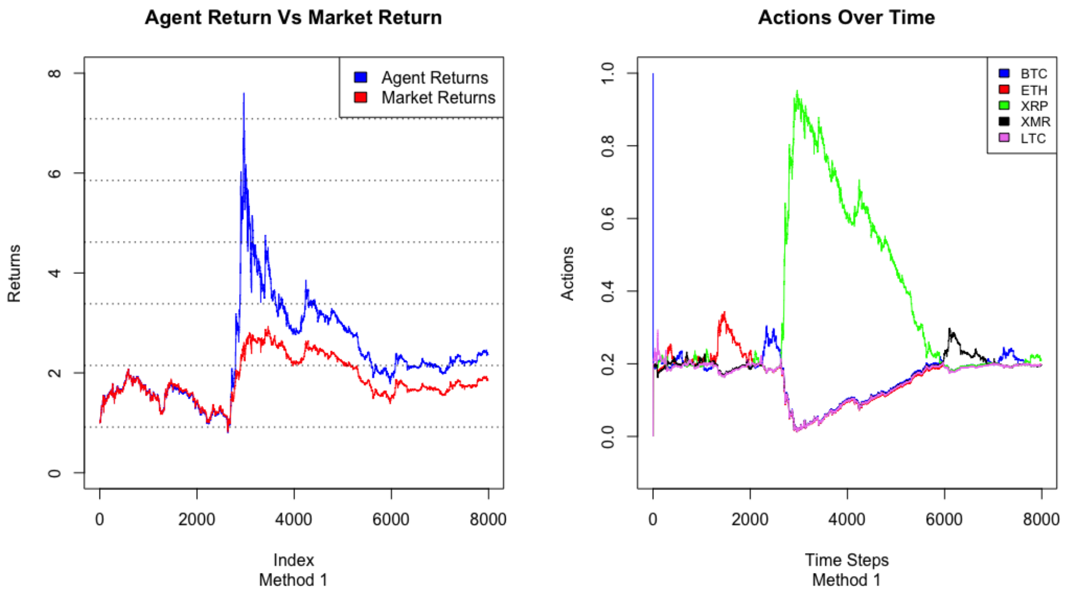

Figure 1: Agent performance and agent actions over time (method 1)

3.2 Method 2

3.2.1 Implementation

Method 2 is another preference function method that takes advantage of the fact that actions carried out by our agent do not affect rewards and transition probabilities. As such it is possible to sample many different actions at each timestep and observe rewards that follow had the agent taken that action.
Again we define a contextual policy function $\pi_t(a|S_t)$ and a normal policy function $\pi_t(a)$ from equations (6) and (13) respectively. We also define a new reward vector R_t^{All} where

$$R_t^{All} := y_t \tag{14}$$

and y_t is the previously defined relative price change vector in (1). This is because the price relative vector is a vector of rewards where the ath element in y_t is the reward resulting from the agent investing all its capital into asset a. This new reward vector R_t^{All} becomes the basis of the action preference updates. In particular, we use a new average reward $\bar{R_t}^{All}$ as

$$\bar{R_t}^{All} := \frac{1}{5t} \sum_{T=1}^{t} \sum_{a=1}^{5} R_{T,a}^{All} \tag{15}$$

where we sum from $a = 1$ to 5 since there are 5 assets in the portfolio and $R^{All}_{T,a}$ denotes the ath element of the vector R^{All}_T. In other words, $\bar{R_t}^{All}$ computes the mean of the concatenation of vectors

$$(R^{All}_1, R^{All}_2, ..., R^{All}_t)^T \tag{16}$$

Intuitively, this means that the average reward $\bar{R_t}^{All}$ becomes the market average at time t. We can then update preferences as follows:

$$H_{t+1}(a) = H_t(a) + \alpha(R^{All}_{t,a} - \bar{R_t}^{All})(1 - \pi_t(a)),\ a = 1, ..., 5 \tag{17}$$

In other words, the preference for each asset is updated according to the observed reward for that asset and the average of all previous rewards, effectively making the method rely on 5 different independent preference functions that monitor one asset each. Like the ad-hoc method, the action taken by the agent is the same as in (9),

$$A'_t = (\pi_t(1|S_t), \pi_t(2|S_t), ..., \pi_t(5|S_t))^T$$

and the reward used to measure its performance is once again

$$R'_{t+1} := y_{t+1} \cdot A'_t$$

However, this time, none of the agent rewards R'_{t+1} are used to update the preference functions $H_t(a)$. This means that the update functions for each $H_t(a)$ are valid since the sequence of preference functions

$$H_1(a), H_2(a), ..., H_t(a)$$

represents the preference values for an agent that invests all its capital into asset a during each trading period (compared to the baseline $\bar{R_t}^{All}$ equal to the market average). The agent's action A'_t is therefore a softmax vector that represents the relative preferences of each indenpendent evaluator. We can predict that method 2's rewards are much less volatile than method 1's since the estimates of each asset are continuously updated at each time step. As such the preferences should also contain less variance.

The function used to simulate a single episode of method 2 is `simulate_contextual2` and the parameters are as follows:

- `n_curren`: The number of currencies in the problem including cash (5 for this project)
- `n_steps`: The number of timesteps T in the episode. To use all available data set `n_steps = nrow(data)`
- `alpha`: Learning rate α when updating action preference function (as defined above)
- `window_size`: Size of historic prices to consider when computing S_t (`window_size` corresponds to the value of n)
- `discount`: Discount factor of previous relative price changes when computing S_t (`discount` corresponds to the value of γ)
- `data`: data that is used to run experiments

3.2.2 Analysis

Using this method the agent follows the market, the returns are slightly below the market but it experiences positive returns (63.8%). The preferences in method 2 varry a lot less than method 1 since the preferences are updated as if each "arm" was pulled (as opposed to the action being the argmax of the softmax output given the state). The largest increase in preference was observed when XRP had a large spike as discussed previously. But the effects were not as drastic as the previous method since the preferences are updated more uniformly. A grid search was once again used to find the optimal parameters as in method one (add reference to grid search results here). In addition, the same range of values were used for alpha and window size. The outcome was that the lowest learning rate would result in the highest return as well as the lowest window size. The main goal of this modeling is to train an agent to beat the market, but in this method it has failed to do so.

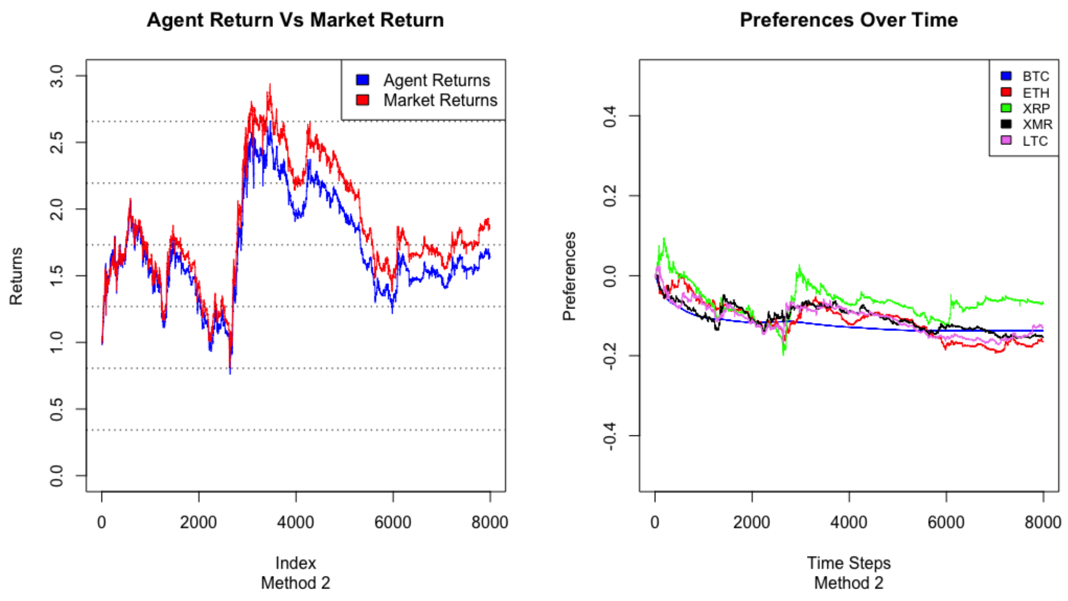

Figure 2: Agent performance and agent preferences over time (method 2)

3.3 Method 3

3.3.1 Implementation

Method 3 (dubbed the "all-or-nothing" method" is the original contextual bandits with action preference function method outlined in the section for method 1. We use the softmax and the previously defined state vector S_t to form our action probabilities $\pi_t(a|S_t)$ and $\pi_t(a)$ ((6) and (13)). The action taken by the agent is then from (7)

$$A_t = \arg\max_a \pi_t(a|S_t)$$

More formally, when computing the reward, the action A_t becomes the standard basis vector w_t where

$$w_t := e_{A_t} \tag{18}$$

and the vectors e are defined as

$$e_1 := (1,0,0,0,0)^T,\ e_2 := (0,1,0,0,0)^T,\ e_3 := (0,0,1,0,0)^T,\ e_4 := (0,0,0,1,0)^T, e_5 := (0,0,0,0,1)^T$$

when the number of assets $k = 5$.

This leads to the aforementioned definition of the reward (8),

$$R_{t+1} := y_{t+1} \cdot A_t = y_{t+1,A_t}$$

which is the relative price change of asset A_t. The preference can now be correctly updated as

$$H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)) \tag{19}$$

$$H_{t+1}(a) = H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a),\ a \neq A_t \tag{20}$$

which differ from (11) and (12) since we are now using the correct observations R_t instead of R'_t.

The all-or-nothing method is essentially the simplest algorithm for tackling the portfolio optimization problem, since it is equivalent to the k-armed contextual bandits framework where we can assume rewards for each arm are non-stationary and we only pull one and one arm only at each time step. In practice, this means that the agent reallocates all the capital of the portfolio into one currency during each trading period based on its preference function, which can lead to extremely volatile rewards as well as large transaction costs (if such fees were considered). In addition, since the we assume non-stationarity, an epsilon greedy strategy is asseumed.

The function used to simulate a single episode of method 3 is `simulate_contextual3` and the parameters are as follows:

- `n_curren`: The number of currencies in the problem including cash (5 for this project)
- `n_steps`: The number of timesteps T in the episode. To use all available data set `n_steps = nrow(data)`
- `alpha`: Learning rate α when updating action preference function (as defined above)
- `window_size`: Size of historic prices to consider when computing S_t (`window_size` corresponds to the value of n)
- `discount`: Discount factor of previous relative price changes when computing S_t (`discount` corresponds to the value of γ)
- `epsilon`: ϵ value (for epsilon greedy)
- `data`: data that is used to run experiments

3.3.2 Analysis

This is the only method that will have an element of randomness, as it uses an epsilon greedy approach. In practice the transaction costs involved in this method would be very high when compared to the other method.The “all or nothing” method experiences the worst returns when compared to the other Contextual Bandit methods. It is the only the method that actually experiences negative returns, but this changes drastically when the seed is changed. The only way this method can perform better is if it is able to identify the currency that will beat the market at each time step, but this is not possible given the simplicity of the model. The grid search was not conducted for this method since the seed changes what the returns will be. This method does not perform well. The non-stationary nature of the problem forces the agent to continuously explore, and thus there is no true convergence.

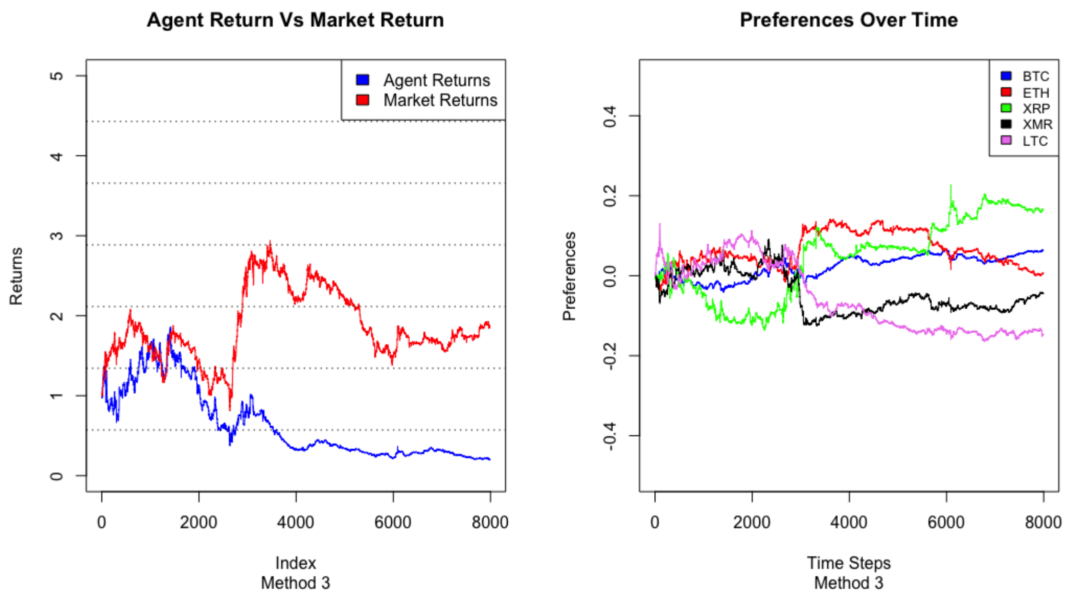

Figure 3: Agent performance and agent preferences over time (method 3)

3.3.3 Contextual bandits comparison

When comparing all contextual bandits methods, it can be seen that the ad-hoc method beats all the other methods as well as outperforming the market. As discussed earlier this is mainly due to the spike in XRP and how the agent decides to heavily invest in it. When only using the data past the spike, the returns from all the methods are very close to each other. It should be noted that method 3 should not be compared it’s previous return since there is randomness and the agent would not make the same decisions it made initially. Method 1 is almost invisible since it has almost the same returns as the market.

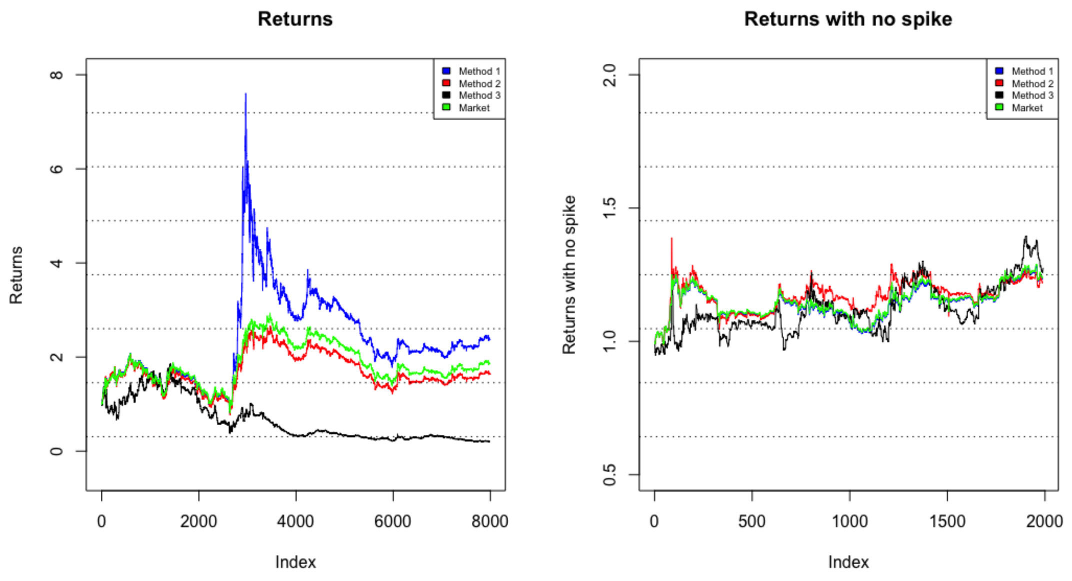

Figure 4: Comparison of performance of first 3 methods (with and without spike)

3.4 Method 4

3.4.1 Implementation

Similarly to method 1, method 4 is also an ad hoc method based on policy gradient method. The portfolio allocation problem is framed as a k-armed bandits problem where the actions are the five cryptocurrencies (including cash). The difference in the two methods comes from the way the action preference is updated. With the policy gradient method the objective is to optimize the preference function. It is defined as a function of some parameters θ. The probability of selecting any action for each state and the transition probability thus also depends on θ. Our implementation closely follows that of Sutton and Barto (add reference here). More specifically, we have

$$\pi_t(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_{b=1}^{5} e^{h(s,b,\theta)}}, \ a = 1, ..., 5 \tag{21}$$

$$h(s,a,\theta) = \theta^T x(s,a) \tag{22}$$

where π denotes once again the preference probabilities and h denotes the preference function. Here the preferences are linear in features ($x(s,a)$ is a diagonal matrix). The parameters θ are also stored in a vector

$$\theta := (\theta_1, ..., \theta_5)^T \tag{23}$$

The parameters are continuously updated and it affects the action selection and the distribution of the states which complicates the calculation of the gradient ascent. There exist a theorem that states that the gradient ascent can be approximated analytically. This theorem allows the calculation of the sample gradient ascent without taking the derivative of the state distribution. This approximation is only proportional to the true value and the learning step size α will have to take into account this constant of proportionality as well. The values of θ are then updated in the following way:

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} \tag{24}$$

where S_t is simply the previous period return and G_t is the current return (i.e. it is a myopic approach to the problem). Note also that

$$\frac{\nabla\pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} = \nabla ln(\pi(A_t|S_t,\theta)) \tag{25}$$

In theory, the agent's action is to put all of the money in the stock with the maximum return. Therefore, A_t and R_{t+1} should be defined as in equations (7) and (8) respectively. However, what the agent truly does is rebalance the portfolio according to the preference function $\pi(a,\theta)$ to obtain

$$R'_{t+1} = y_{t+1} \cdot (\pi(1,\theta), \pi(2,\theta), \pi(3,\theta), \pi(4,\theta), \pi(5,\theta))^T \tag{26}$$

thus making this an ad hoc method akin to method 1. Method 4 is therefore an alternative implementation of method 1, where the agent behaves myopically. We rely again on the simplifying assumption that

$$E[R_t] \approx E[R'_t]$$

and we can expect results to be similar to a myopic implementation of method 1.

The function used to simulate a sing episode of method 4 is `GradPolControl` and the parameters are as follows:

- `init_weights`:
- `theSeed`:
- `length_Epis`: Length of episode
- `alphasteptheta`:
- `startingpointepisode`: Initial starting point in data of the episode (defaults to 0)

3.4.2 Analysis

Intuitively, it is not safe to make the assumption that $E[R_t] \approx E[R_{t+1}]$ since the action of the agent is different. The results are however encouraging. For episodes with 200 periods the agent beats the market about 63% of the time. Interestingly, the step size made no difference in the percentage of time the agent beats the market. One could conclude that the learning step size makes does not matter, but clearly by looking at the graph comparing the market return and the various step size, it is safe to conclude that the learning step size matters. The weights of each cryptocurrency in the portfolio varied a lot more when there was a smaller learning step, thus there was a difference in the return. The timing at which the episode make a big difference in the performance of the agent. The agent seems to be performing much better when the episodes starts after the spike.

As expected the agent performs a lot better when the learning step size is smaller (i.e. the agent adapts faster to the change in the market). The fact that the agent mimics pretty closely what the market does is also not surprising since the gradient re-adjust the weights of the currencies depending on the direction the market is taking. The portfolio unexpectedly falls well below the baseline value (i.e. Bitcoin). One would expect the agent to reallocate the cash towards the "safe" choice. The baseline with a constant return of zero is the best choice in this case. From the graph below there

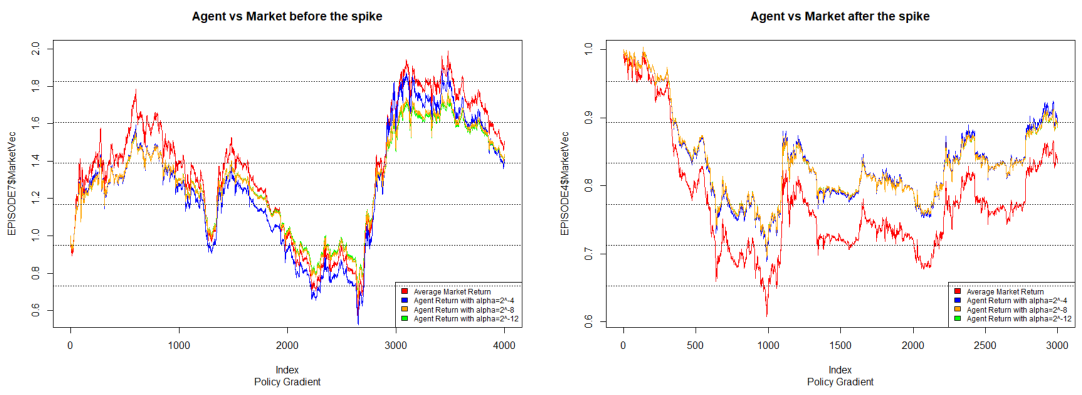

Figure 5: Policy gradient performance before and after spike

is no increase in the bitcoin preference. One way to interpret this is to think that the agent would rather go towards something more volatile since there is more chance of a higher return. It is a major drawback of this method because high risk can lead to high return but to substantial losses as well. However, the biggest drawback of this method is obviously its lack of mathematical foundation. It is not mathematically sound.

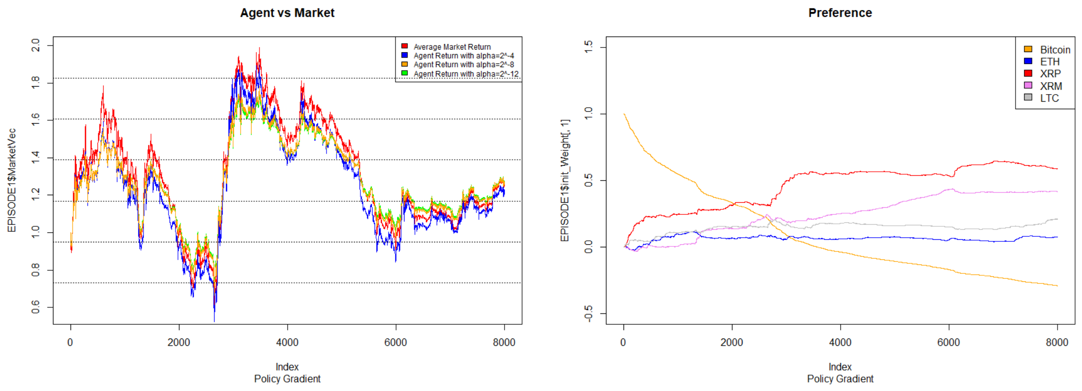

Figure 6: Agent performance and agent preferences over time (policy gradient)

4 Conclusion

Conclusion goes here

5 Appendix

A Figures

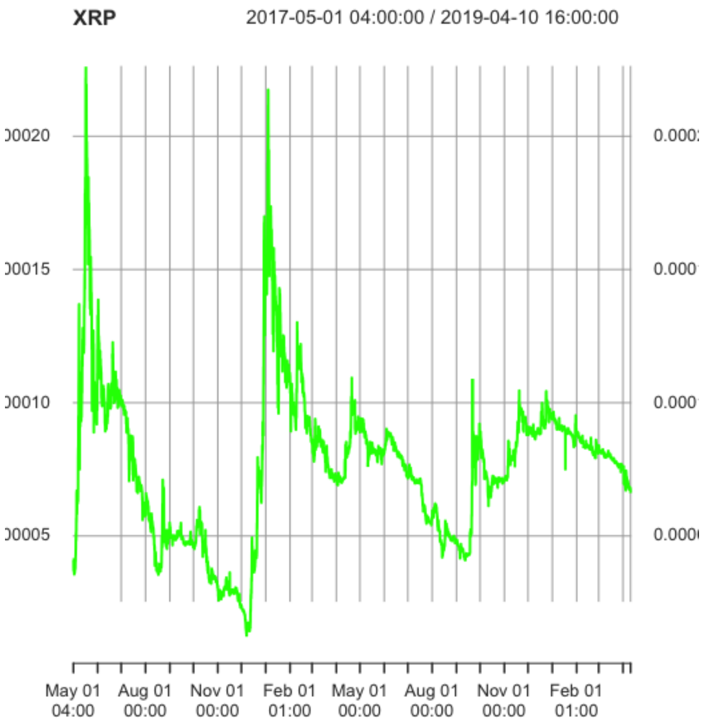

Figure A.1: Observed spike in XRP

$\alpha\backslash WindowSize$	6	7	8	9	10	11	12
0.1	1.855655	1.903908	1.833427	1.722254	1.761848	1.731211	1.809517
0.15	1.998127	1.730006	1.720069	2.001871	1.982850	1.702941	1.727604
0.20	2.027906	1.880474	1.678368	1.876704	1.689923	1.648864	1.723446
0.25	1.661065	2.028314	2.093625	1.650993	2.141800	1.569624	1.981973
0.30	1.963076	2.140941	1.973012	1.615982	2.163592	1.570967	1.676110
0.35	1.649652	1.564450	1.520341	2.180393	1.974001	1.464699	2.308119
0.40	2.299351	1.997830	2.007928	1.587208	1.476948	2.287297	**2.365864**

Figure A.2: Grid search results for contextual 1

$\alpha \backslash WindowSize$	6	7	8	9	10	11	12
0.1	**1.638191**	1.572772	1.5660357	1.5514709	1.5737719	1.5522290	1.5949086
0.15	1.538498	1.466717	1.4555418	1.4368205	1.4544121	1.4310132	1.4691287
0.20	1.441293	1.364100	1.3488002	1.3264458	1.3397009	1.3148013	1.3485967
0.25	1.347864	1.266357	1.2475844	1.2223093	1.2319287	1.2060082	1.2360816
0.30	1.259554	1.174910	1.1536115	1.1262614	1.1332199	1.1068277	1.1340770
0.35	1.177614	1.091010	1.0683215	1.0397886	1.0452239	1.0189028	1.0444117
0.40	1.103089	1.015601	0.9927195	0.9638487	0.9689299	0.9431457	0.9680506

Figure A.3: Grid search results for contextual 2

B References

References go here