DEEP OPTION HEDGING USING RISK-AVERSE CONTEXTUAL BANDIT LEARNING

MSBA 7021 | Group 40
3035878248 | Li, Wenhao
3035886336 | Sun, Zhiting

Outline

01 Intro & Literature Review

What is Option Hedging; DL / ML in Option Hedging; Objective Function

02 Data Simulation & Model Design

Methodology & Experiments

03 Analysis

Empirical Results and Discussion

04 Conclusions

Key Findings and Limitations

01

Intro & Literature Review

What is Option Hedging; DL / ML in Option Hedging; Objective Function

What is Option Hedging I

Motivation

Protect against **market volatility**

To reduce, or hedge, the directional risk associated with price movements in the underlying assets

Option Hedging:

- A strategy used to reduce the risk of an investment
- Involves the use of **derivatives** to offset potential losses

Two Problems of Continuous Replication

UNREALISTIC

Complete and frictionless markets do not exist

UNDESIRABLE

High transaction costs

Such as the **Black, Scholes and Merton (BSM) Model** that is widely used to price options contracts.

What is Option Hedging II

Formulation

Profit & Loss

- the cost of hedging is calculated **period-by-period** as the change in the value of the hedged position (option plus stock) plus the trading costs associated with changing the position in the stock.
- requires a pricing model

Cash Flow

- the costs are the **cash outflows** and **inflows** from trading the stock and there is a potential final negative cash flow payoff on the option

Approach

Utility-Based

- where a portfolio's performance is measured by **expected utility**
- a multi-period planning problem, whose goal is the optimal trade-off between risk and global transaction costs

Risk-Averse

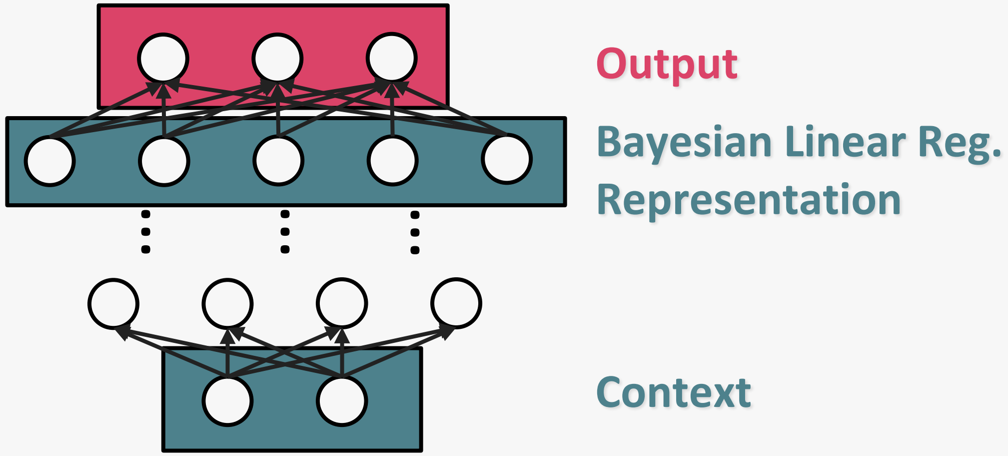
- Extending on the utility-based approach, add **risk terms** and choose a **mean-variance reward function** to minimize losses

Deep Learning / Machine Learning in Option Hedging I

Reinforcement Learning — **Contextual Multi-armed Bandit (CMAB)**

Main Model

- Bayesian Neural Network Risk-Averse CMAB



Alternatives

- Neural Greedy Bandits
- Naïve Bandits
- Random Bandits
- Predict-Then-Optimize with SARIMAX
- Oracle (Best-Case Scenario)

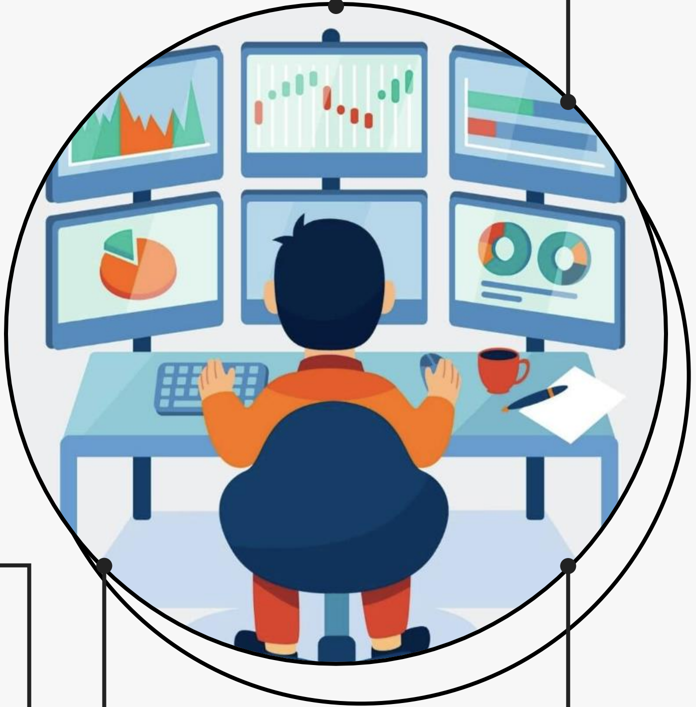
DL/ ML in Option Hedging II

- Can incorporate **prior knowledge** into the model
- Can estimate **uncertainty** in the parameters
- Can handle missing data and outliers

Advantages of BLR

Bayesian Linear Regression

- Uses **Bayesian inference** to calculate the posterior distribution of the parameters
- Assumes a prior



Thompson Sampling

- A **probabilistic** algorithm used in reinforcement learning
- Uses Bayesian inference to select the best action
- Exploits the uncertainty of the environment

Comparison between Thompson Sampling and Epsilon-Greedy

Thompson Sampling	Epsilon-Greedy
Probabilistic	Deterministic
Exploits **uncertainty**	Exploits **certainty**
More efficient	

Objective Function, Cost & Portfolio Value

Hedging Portfolio Value

$$\Pi_t = n_t S_t + B_t \quad (\text{self-financing})$$

$$\begin{aligned}\Delta\Pi_{t+1} &= n_{t+1} S_{t+1} + \Delta B_{t+1} - n_t S_t + Cost(n_t, n_{t+1}, S_t) \\ &= n_{t+1}\Delta S_{t+1} + Cost(n_t, n_{t+1}, S_t)\end{aligned}$$

Transaction Cost

$$Cost(n_t, n_{t+1}, S_t) = -\eta \cdot S_t \cdot |n_{t+1} - n_t|$$

Delta Wealth

$$k \cdot E[-\Delta C_{t+1} + n_{t+1}\Delta S_{t+1} + Cost(n_t, n_{t+1}, S_t) \mid F_t]$$

Risk Term

$$V[-\Delta C_{t+1} + n_{t+1}\Delta S_{t+1} + Cost(n_t, n_{t+1}, S_t) \mid F_t]$$

Utility / Reward

$$u_t = \underline{k \cdot E[-\Delta C_{t+1} + n_{t+1}\Delta S_{n+1} + Cost(n_t, n_{t+1}, S_t) \mid F_t]} - \underline{V[-\Delta C_{t+1} + n_{t+1}\Delta S_{n+1} + Cost(n_t, n_{t+1}, S_t) \mid F_t]}$$

Formulation	Approaches
● Profit & Loss	Utility - Based
Cash Flow	● Risk - Averse

02

Data Simulation & Model Design

Methodology & Experiments

Data Simulation

Stock Price

Geometric Brownian Motion (GBM)

- a continuous-time stochastic process in which the logarithm of the randomly varying quantity follows a Brownian motion with drift.

$$f_{S_t}(s;\mu,\sigma,t) = \frac{1}{\sqrt{2\pi}}\frac{1}{s\sigma\sqrt{t}}\exp\left(-\frac{\left(\ln s - \ln S_0 - \left(\mu - \frac{1}{2}\sigma^2\right)t\right)^2}{2\sigma^2 t}\right)$$

$$S_t = S_0\exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right),\quad W_t = W_t - W_0 \sim N(0,t)$$

Option Price

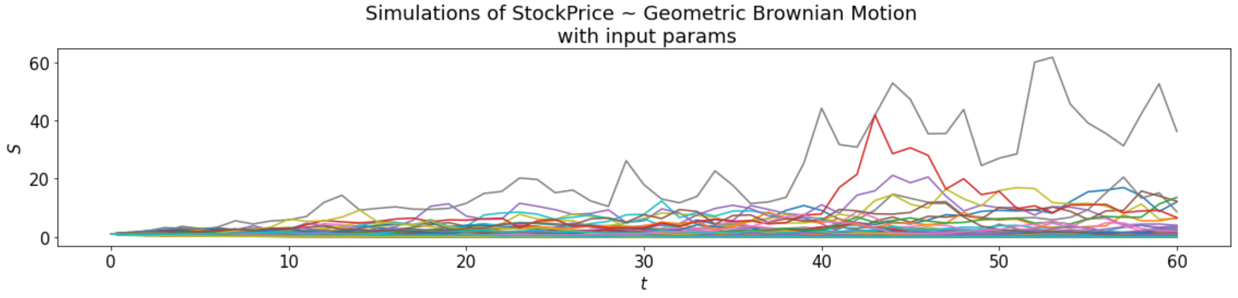
Black-Scholes Model (BSM)

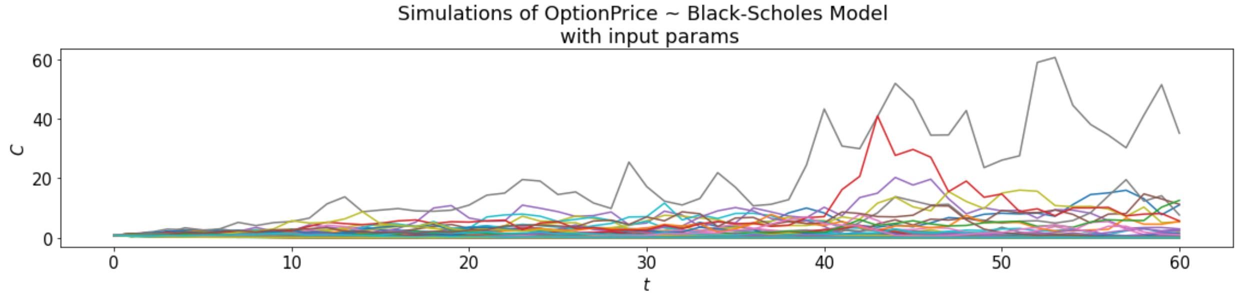
- a pricing model used to determine the fair price or theoretical value for a call or a put option.

$$C(S_t,t) = N(d_1)S_t - N(d_2)Ke^{-r(T-t)}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$





Algorithms

Utility Function

$$u_t = k \cdot E[-\Delta C_{t+1} + n_{t+1}\Delta S_{n+1} + Cost(n_t, n_{t+1}, S_t) \mid F_t] - V[-\Delta C_{t+1} + n_{t+1}\Delta S_{n+1} + Cost(n_t, n_{t+1}, S_t) \mid F_t]$$

Predict-Then-Optimize

Algorithm: Predict-Then-Optimize Algorithm

Predict

for *episode* = 1 : *number of episodes* **do**
  for t = 1 : 5 **do**
    $S_{episode,t}$+= random.uniform(-1, 1)
  for t = 5 : *length(episode)* **do**
    Fit SARIMA model on train data $S_{episode}[:t]$
    Predict *StockPrice* at time t+1 and get $pred\_S_{t+1}$

Optimize

for *episode* = 1 : *number of episodes* **do**
  for t = 1 : *length(episode)-1* **do**
    for a_t = -1, -0.96, …, 1 **do**
      Compute a_t on predicted *OptionPrice*
      Observe predicted reward $pred\_r_t$ and reward r_t
    Select best a_t with highest $pred\_r_t$

Bayesian NN R-CMAB

Algorithm 1: Neural R-CMAB Algorithm

Input Init. action-value NN with random parameters; Set up prior distribution over models, $\pi_0 : \theta \in \Theta \to [0, 1]$. Init. replay memory to given capacity;
for *episode* $= 1 : numberOfEpisodes$ **do**
  for $t = 1 : length(episode)$ **do**
    Sample parameters $\theta_t \sim \pi_t$;
    Get a context vector $x_t \in \mathbb{R}^d$;
    Compute $a_t = \text{BestAction}(x_t, \theta_t)$ through NN followed by linear Bayesian model;
    Perform action a_t on hedging environment and observe reward r_t;
    Add (x_t, a_t, r_t) to memory buffer;
    if *updateModelFrequencyCriterium* **then**
      Update Bayesian estimate of the posterior distribution to π_{t+1} using most recent transitions;
    end
    if *updateNNFrequencyCriterium* **then**
      Sample random batches from memory buffer to train NN;
    end
  end
end

Thompson Sampling & BLR

Algorithm 1 Thompson Sampling

1: *Input:* Prior distribution over models, $\pi_0 : \theta \in \Theta \to [0, 1]$.
2: **for** time $t = 0, \ldots, N$ **do**
3:   Observe context $X_t \in \mathbb{R}^d$.
4:   Sample model $\theta_t \sim \pi_t$.
5:   Compute $a_t = \text{BestAction}(X_t, \theta_t)$.
6:   Select action a_t and observe reward r_t.
7:   Update posterior distribution π_{t+1} with (X_t, a_t, r_t).

Algorithm: Bayesian Linear Regression

The posterior at time t for action i, after observing X, Y, is $\pi_t(\beta, \sigma^2) = \pi_t(\beta \mid \sigma^2)\, \pi_t(\sigma^2)$, where we assume $\sigma^2 \sim \text{IG}(a_t, b_t)$, and $\beta \mid \sigma^2 \sim \mathcal{N}(\mu_t, \sigma^2\, \Sigma_t)$, an Inverse Gamma and Gaussian distribution, respectively. Their parameters are given by

$$\Sigma_t = \left(X^T X + \Lambda_0\right)^{-1}, \qquad \mu_t = \Sigma_t \left(\Lambda_0 \mu_0 + X^T Y\right), \tag{1}$$

$$a_t = a_0 + t/2, \qquad b_t = b_0 + \frac{1}{2}\left(Y^T Y + \mu_0^T \Sigma_0 \mu_0 - \mu_t^T \Sigma_t^{-1} \mu_t\right). \tag{2}$$

We set the prior hyperparameters to $\mu_0 = 0$, and $\Lambda_0 = \lambda\, \text{Id}$, while $a_0 = b_0 = \eta > 1$. It follows that initially, for $\sigma_0^2 \sim IG(\eta, \eta)$, we have the prior $\beta \mid \sigma_0^2 \sim \mathcal{N}(0, \sigma_0^2/\lambda\, \text{Id})$, where $\mathbf{E}[\sigma_0^2] = \eta/(\eta - 1)$. Note that we independently model and regress each action's parameters, β_i, σ_i^2 for $i = 1, \ldots, k$.

Random Bandits

Oracle

Naïve Bandits

Neural Greedy Bandits

Data Buffer

Able to:

- Keep metadata, e.g., num_actions, buffer_size…
- add context-action-reward triplet at each step
- Perform mean/std scaling on context (feature) data
- sample minibatch for neural network training
- return observations for an action to update BLR
- fit intercept to BLR

MSBA7021Project.ContextActionRewardDataset

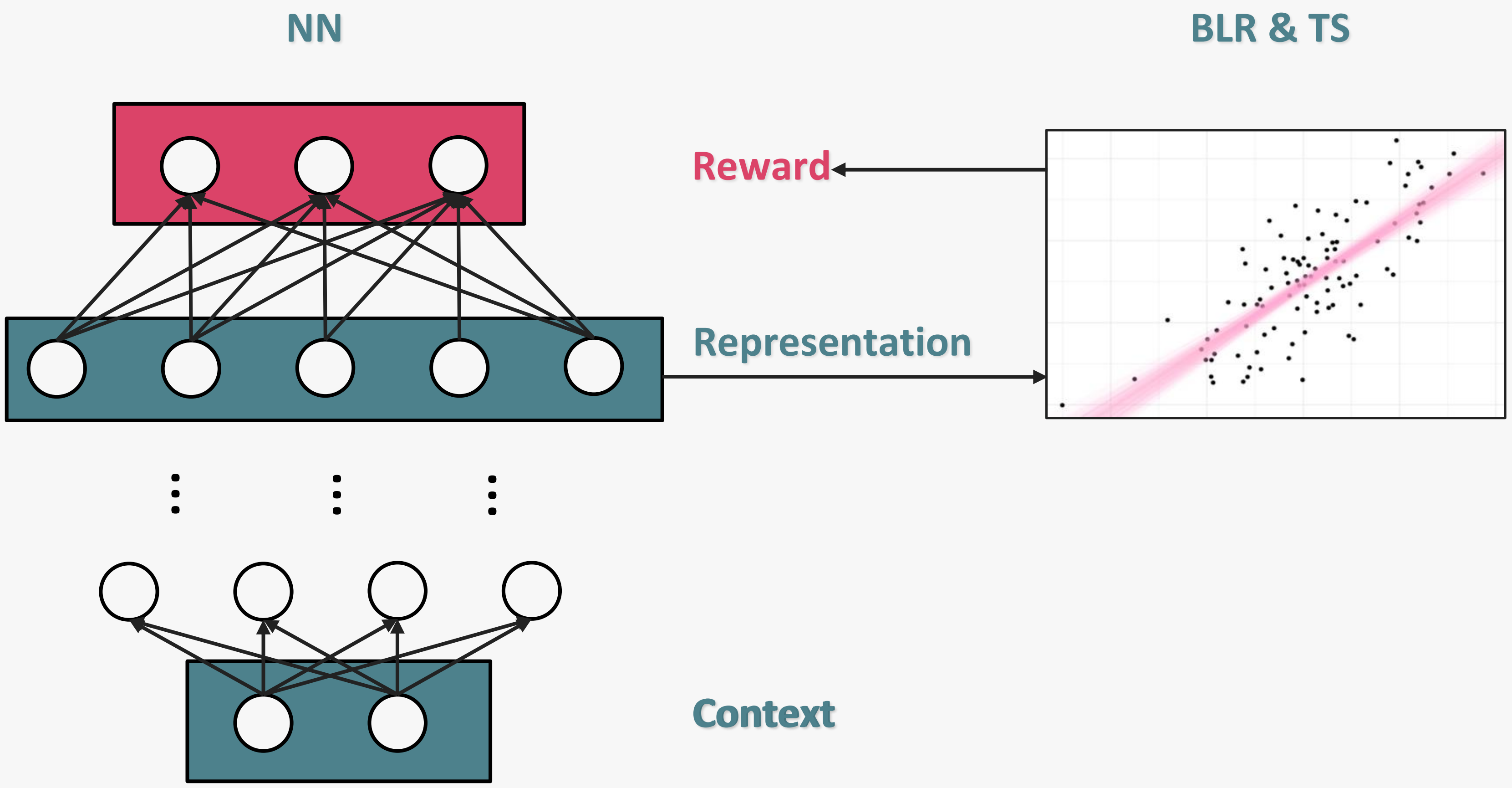
```
_num_actions:
_actions:
scaled_contexts:
_context_dim:
intercept:
_contexts:
_scaled_contexts:
buffer_size:
contexts:
actions:
rewards:
_rewards:
```

```
__init__(self, context_dim, num_actions, buffer_size=-1, intercept=False):
add(self, context, action, reward):
scale_contexts(self, contexts=None):
replace_data(self, contexts=None, actions=None, rewards=None):
get_batch(self, batch_size):
get_data(self, action):
get_data_with_weights(self):
get_batch_with_weights(self, batch_size, scaled=False):
get_contexts(self, scaled=False):
num_points(self, f=None):
context_dim(self):
num_actions(self):
contexts(self):
contexts(self, value):
actions(self):
actions(self, value):
rewards(self):
rewards(self, value):
scaled_contexts(self):
scaled_contexts(self, value):
```

Hedging Implementation

Learning Agents

Bayesian Neural Bandit



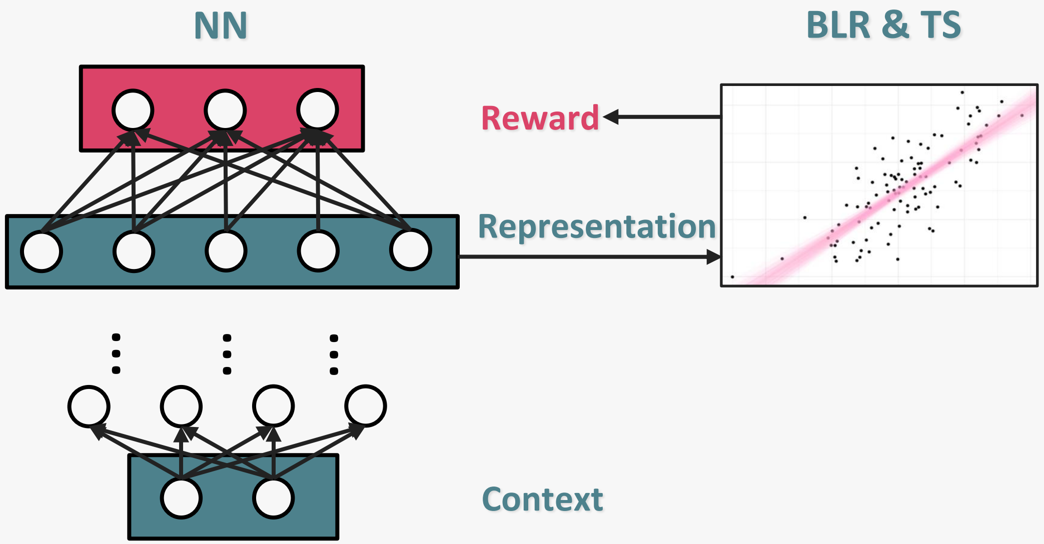
MSBA7021Project.BayesianNeuralBandit

a:
b:
data_h:
update_freq_bayesian:
latent_h:
net_times_trained:
precision:
mu:
update_freq_nn:
cov:
num_actions:
_b0:
_lambda_prior:
_a0:
loss:
t:
optimizer:
context_dim:
net:
hparams:
latent_dim:
num_epochs:
max_grad_norm:
get_net:

__init__(self, **kwargs):
predict(self, context):
action(self, context):
update(self, context, action, reward):
a0(self):
b0(self):
lambda_prior(self):
data(self):

Learning Agents

Bayesian Neural Bandit



Algorithm 1: Neural R-CMAB Algorithm

Input Init. action-value NN with random parameters; Set up prior distribution over models, $\pi_0 : \theta \in \Theta \to [0, 1]$. Init. replay memory to given capacity;

for $episode = 1 : numberOfEpisodes$ **do**
    for $t = 1 : length(episode)$ **do**
        Sample parameters $\theta_t \sim \pi_t$;
        Get a context vector $x_t \in \mathbb{R}^d$;
        Compute $a_t = \mathrm{BestAction}(x_t, \theta_t)$ through NN followed by linear Bayesian model;
        Perform action a_t on hedging environment and observe reward r_t;
        Add (x_t, a_t, r_t) to memory buffer;
        if *updateModelFrequencyCriterium* **then**
            Update Bayesian estimate of the posterior distribution to π_{t+1} using most recent transitions;
        end
        if *updateNNFrequencyCriterium* **then**
            Sample random batches from memory buffer to train NN;
        end
    end
end

Learning Agents

Bayesian Neural Bandit

Algorithm 1: Neural R-CMAB Algorithm

Input Init. action-value NN with random parameters; Set up prior distribution over models, $\pi_0 : \theta \in \Theta \to [0, 1]$. Init. replay memory to given capacity;

for $episode = 1 : numberOfEpisodes$ **do**

  for $t = 1 : length(episode)$ **do**

    Sample parameters $\theta_t \sim \pi_t$;

    Get a context vector $x_t \in \mathbb{R}^d$;

    Compute $a_t = \text{BestAction}(x_t, \theta_t)$ through NN followed by linear Bayesian model;

    Perform action a_t on hedging environment and observe reward r_t;

    Add (x_t, a_t, r_t) to memory buffer;

    if *updateModelFrequencyCriterium* **then**

      Update Bayesian estimate of the posterior distribution to π_{t+1} using most recent transitions;

    end

    riterium **then**

    hes from memory buffer to train NN;

The posterior at time t for action i, after observing X, Y, is $\pi_t(\beta, \sigma^2) = \pi_t(\beta \mid \sigma^2)\, \pi_t(\sigma^2)$, where we assume $\sigma^2 \sim \text{IG}(a_t, b_t)$, and $\beta \mid \sigma^2 \sim \mathcal{N}(\mu_t, \sigma^2\, \Sigma_t)$, an Inverse Gamma and Gaussian distribution, respectively. Their parameters are given by

$$\Sigma_t = \left(X^T X + \Lambda_0\right)^{-1}, \qquad \mu_t = \Sigma_t \left(\Lambda_0 \mu_0 + X^T Y\right), \tag{1}$$

$$a_t = a_0 + t/2, \qquad b_t = b_0 + \frac{1}{2}\left(Y^T Y + \mu_0^T \Sigma_0 \mu_0 - \mu_t^T \Sigma_t^{-1} \mu_t\right). \tag{2}$$

We set the prior hyperparameters to $\mu_0 = 0$, and $\Lambda_0 = \lambda\, \text{Id}$, while $a_0 = b_0 = \eta > 1$. It follows that initially, for $\sigma_0^2 \sim IG(\eta, \eta)$, we have the prior $\beta \mid \sigma_0^2 \sim \mathcal{N}(0, \sigma_0^2/\lambda\, \text{Id})$, where $\mathbf{E}[\sigma_0^2] = \eta/(\eta - 1)$. Note that we independently model and regress each action's parameters, β_i, σ_i^2 for $i = 1, \ldots, k$.

Learning Agents

Bayesian Neural Bandit

Algorithm 1: Neural R-CMAB Algorithm

Input Init. action-value NN with random parameters; Set up prior distribution over models, $\pi_0 : \theta \in \Theta \to [0, 1]$. Init. replay memory to given capacity;

for *episode* $= 1 :$ *numberOfEpisodes* **do**
- **for** $t = 1 : length(episode)$ **do**
  - Sample parameters $\theta_t \sim \pi_t$;
  - Get a context vector $x_t \in \mathbb{R}^d$;
  - Compute $a_t = \text{BestAction}(x_t, \theta_t)$ through NN followed by linear Bayesian model;
  - Perform action a_t on hedging environment and observe reward r_t;
  - Add (x_t, a_t, r_t) to memory buffer;
  - **if** *updateModelFrequencyCriterium* **then**
    - Update Bayesian estimate of the posterior distribution to π_{t+1} using most recent transitions;
  - **end**
  - **if** *updateNNFrequencyCriterium* **then**
    - Sample random batches from memory buffer to train NN;
  - **end**

Neural Network Architectures All algorithms based on neural networks as function approximators share the same architecture. In particular, we fit a simple fully-connected feedforward network with 3 hidden layers with 20 units each and ReLu activations. The input of the network has dimension d (same as the contexts), and there are k outputs, one per action. Note that for each training point (X_t, a_t, r_t) only one action was observed (and algorithms usually only take into account the loss corresponding to the prediction for the observed action).

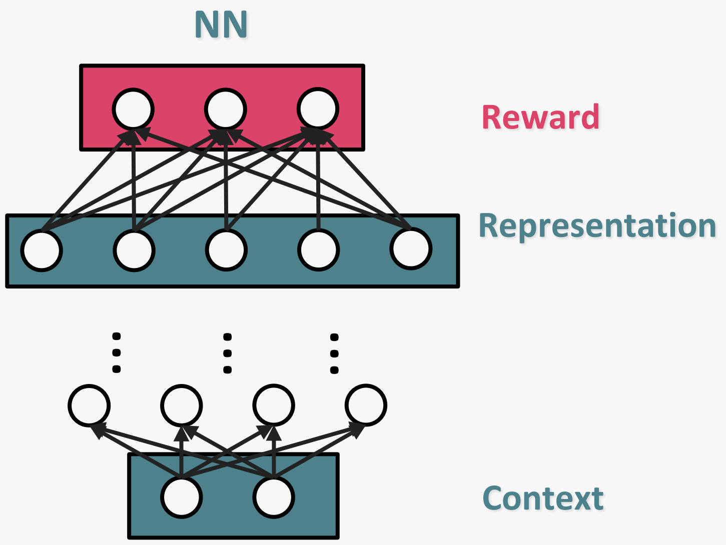
```
Signature:
ContextActionRewardDataset.get_batch_with_weights(
    self,
    batch_size,
    scaled=False,
)
Docstring: Return a random mini-batch with one-hot weights for actions
```

```
array([[-0.00778823,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        , -0.23062548,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        , -0.21748361, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.06482949,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.19679239,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [-0.2201423 ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ]])
```

Hedging Implementation

Learning Agents

Neural Greedy Bandit



MSBA7021Project.NeuralGreedyBandit

data_h:
net_times_trained:
update_freq_nn:
num_actions:
epsilon:
loss:
t:
optimizer:
context_dim:
net:
hparams:
num_epochs:
max_grad_norm:
get_net:

__init__(self, **kwargs):
predict(self, context):
action(self, context):
update(self, context, action, reward):
data(self):

Algorithm: Neural Greedy Bandit Algorithm

Input Init. action-value NN with random parameters. Init. replay memory to given capacity;

for *episode* = 1 : *number of episodes* **do**
    for t = 1 : *length(episode)* **do**
        Get a context vector x_t
        Compute a_t= BestAction(x_t, θ_t) through NN
        Perform action a_t on hedging environment and observe reward r_t;
        Add (x_t, a_t, r_t) to memory buffer;
        if updateNNFrequencyCriteron **then**
            Sample random batches from memory buffer to train NN;
        end
    end
end

Learning Agents

Naïve Bandit

MSBA7021Project.NaiveBandit
epsilon: data_h: t: context_dim: num_actions: hparams:
__init__(self, **kwargs): predict(self, context): action(self, context): update(self, context, action, reward): data(self):

- Vanilla ε-greedy MAB

Random Bandit

MSBA7021Project.RandomBandit
data_h: t: context_dim: num_actions: hparams:
__init__(self, **kwargs): action(self, context): update(self, context, action, reward): predict(self, context): data(self):

- Random pulls, i.e., pure exploration

Learning Agents

Predict-Then-Optimize SARIMAX

Algorithm: Predict-Then-Optimize Algorithm

Predict

for *episode* = 1 : *number of episodes* **do**
    for $t = 1 : 5$ **do**
        $S_{episode,t}$ += random.uniform(-1, 1)
    for $t = 5$: *length(episode)* **do**
        Fit SARIMA model on train data $S_{episode}[:t]$
        Predict *StockPrice* at time t+1 and get $pred\_S_{t+1}$, $pred\_C_{t+1}$,

Optimize

for *episode* = 1 : *number of episodes* **do**
    for $t = 1$: *length(episode)* **do**
        for a_t = -1, -0.96, …, 1 **do**
            Compute a_t on predicted *OptionPrice*
            Observe predicted reward $pred\_r_t$
        Select best a_t with highest $pred\_r_t$
        Use a_t and observe reward r_t

Trainer Function

```
Signature: trainBandit(model_cls, num_episodes, T, S, C, test_func, **kwargs)
Docstring: Construct-trains a bandit with model-specific input arguments, and retu
```

```
hedgeBandit_bayesianNeural, \
train_episodic_cumRewards_bayesianNeural, \
test_episodic_cumRewards_bayesianNeural, \
oracle_episodic_cumRewards_bayesianNeural, \
oracle_actions_bayesianNeural = trainBandit(
                            BayesianNeuralBandit,
                            num_episodes,
                            T=T,
                            S=S,
                            C=C,
                            test_func=testBandit,
                            num_actions=51,
                            context_dim=2,
                            latent_dim=20,
                            get_net=get_net,
                            learning_rate=0.1,
                            max_grad_norm=10.0,
                            lr_decay=1,
                            batch_size=512,
                            weight_decay=0,
                            show_training=False,
                            buffer_size=-1,
                            initial_pulls=10,
                            training_freq_baysian=1,
                            training_freq_network=50,
                            training_epochs=100,
                            do_scaling=False,
                            a0=6,
                            b0=6,
                            lambda_prior=0.25,
                            exploration_rate=0.3,
                            verbose=False,
                            summary_per_epoch=50)
```

Bayesian Optimization Hyperparams Tuning

def *objectiveFuncBO*(params):

declare training variables

simulate data

trainBandit(simulatedData, params)

return meanEpisodicCumReward

Iteration	Y	var_1	var_2
1.0	158.64480596634627	1e-05	5.0
2.0	72.81709722930584	0.1	50.0
3.0	42.73818185596484	10.0	50.0
4.0	35.529717832797544	10.0	10.0
5.0	254.39865395791574	0.001	10.0
6.0	133.19784284668012	1e-05	10.0
7.0	73.69479617706655	0.001	10.0
8.0	32.48165280446281	10.0	5.0
9.0	52.481452647299456	10.0	5.0
⋮			

...

```
# define hyperparams bounds
bds = [
    {'name': 'learning_rate', 'type': 'discrete', 'domain': (0.00001, 0.001, 0.1, 10.0)},
    {'name': 'max_grad_norm', 'type': 'discrete', 'domain': (5.0, 10.0, 50.0)},
    {'name': 'lr_decay', 'type': 'discrete', 'domain': (1, 5)},
    {'name': 'batch_size', 'type': 'discrete', 'domain': (128, 512, 1024)},
    {'name': 'initial_pulls', 'type': 'discrete', 'domain': (5, 20)},
    {'name': 'training_freq_network', 'type': 'discrete', 'domain': (50, 100, 150)},
    {'name': 'training_epochs', 'type': 'discrete', 'domain': (50, 100)},
    {'name': 'do_scaling', 'type': 'discrete', 'domain': (True, False)}
]
```

03

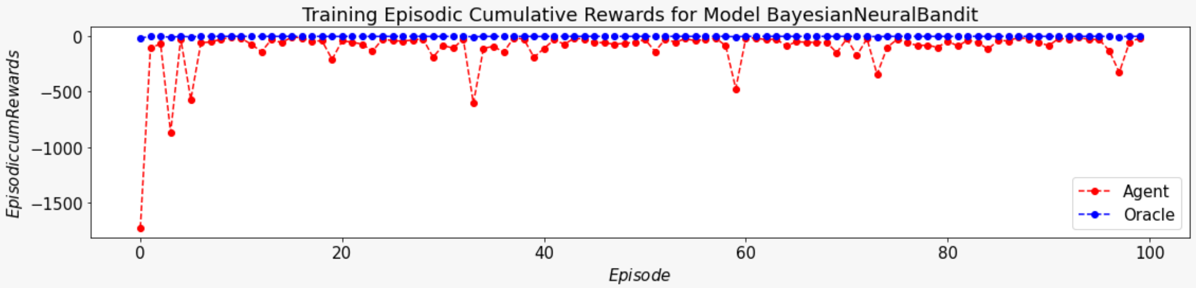
Analysis

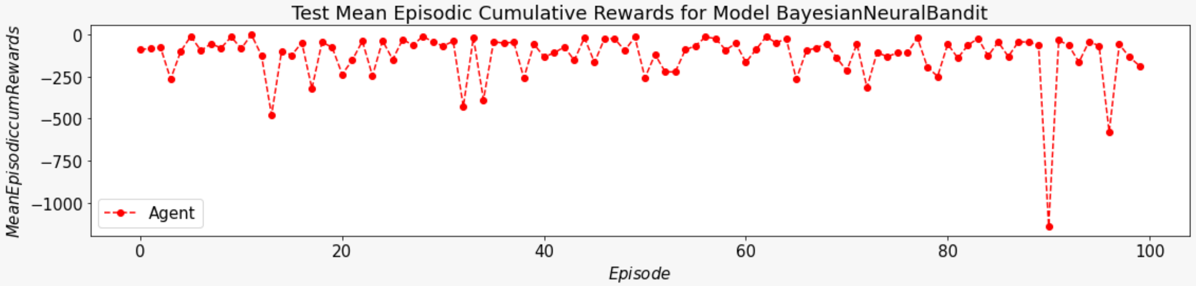
Empirical Results and Discussion

Training Process and Test Performance

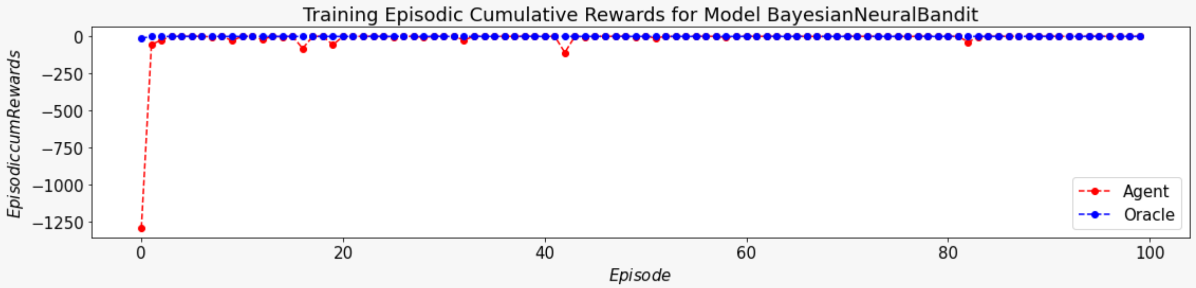
Bayesian Neural Bandit

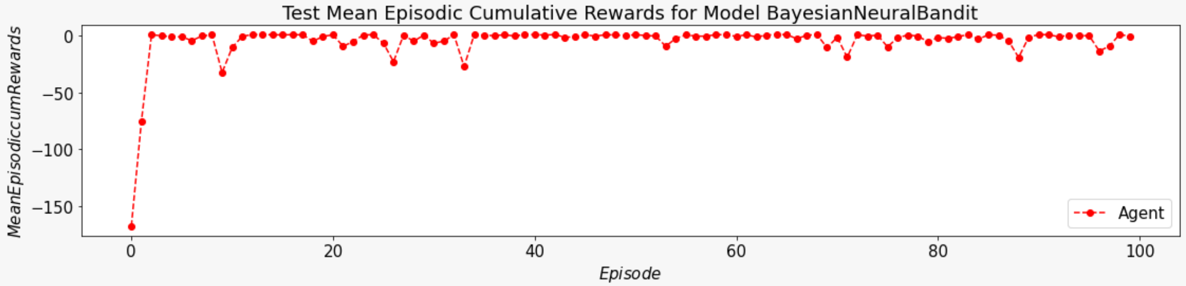
w. short-selling





w.o. short-selling



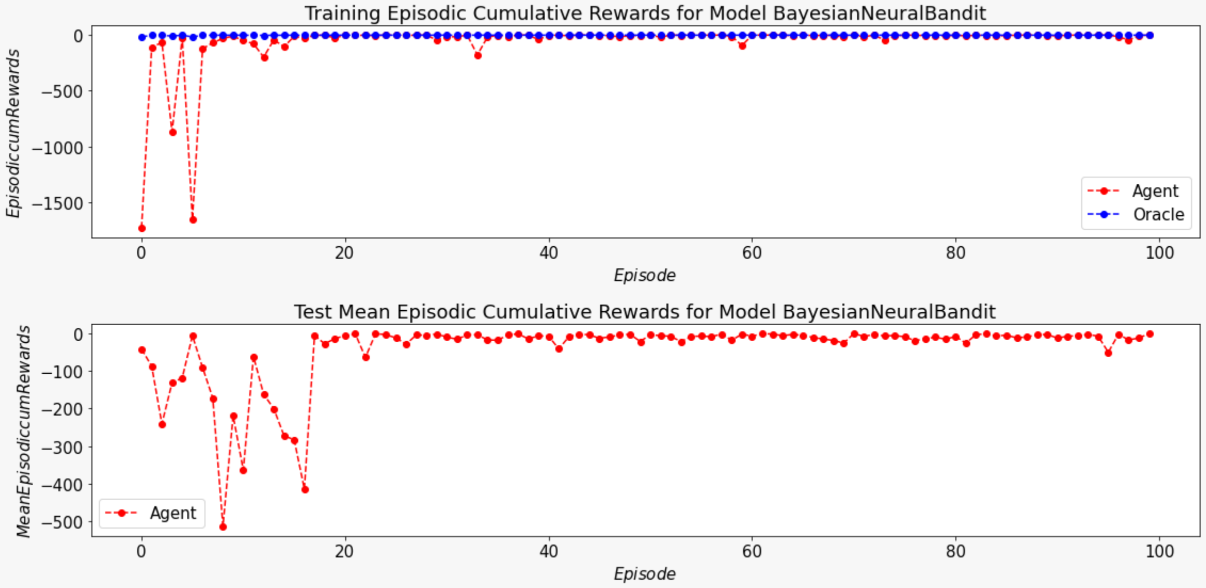


*Test performance based on 10 episodes of unseen data

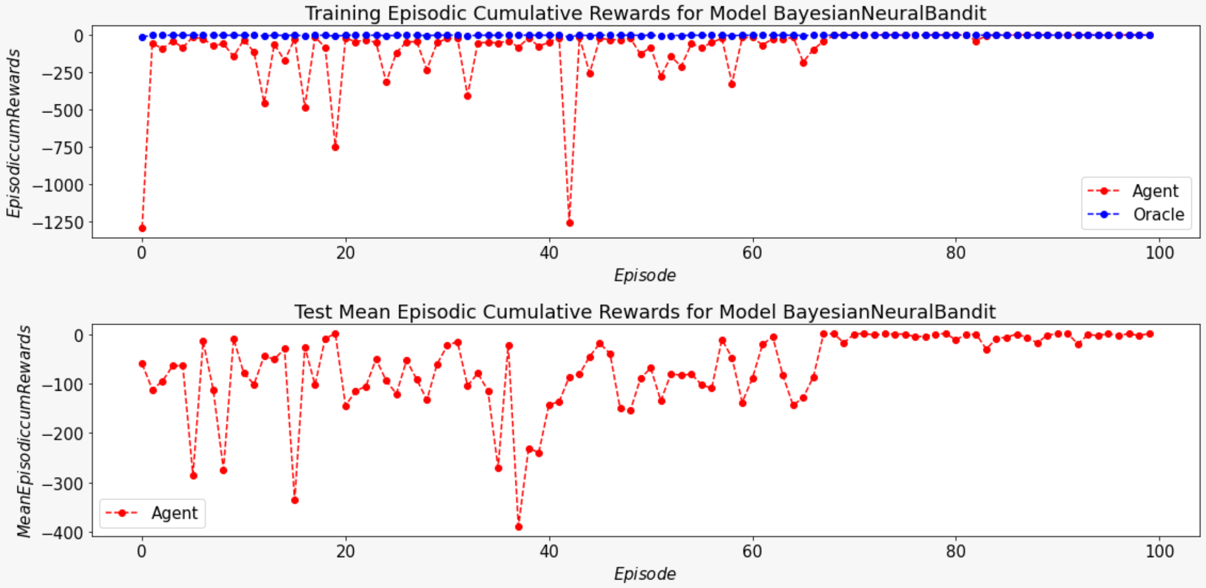
Training Process and Test Performance

Bayesian Neural Bandit with Tuned Hyperparams

w. short-selling



w.o. short-selling



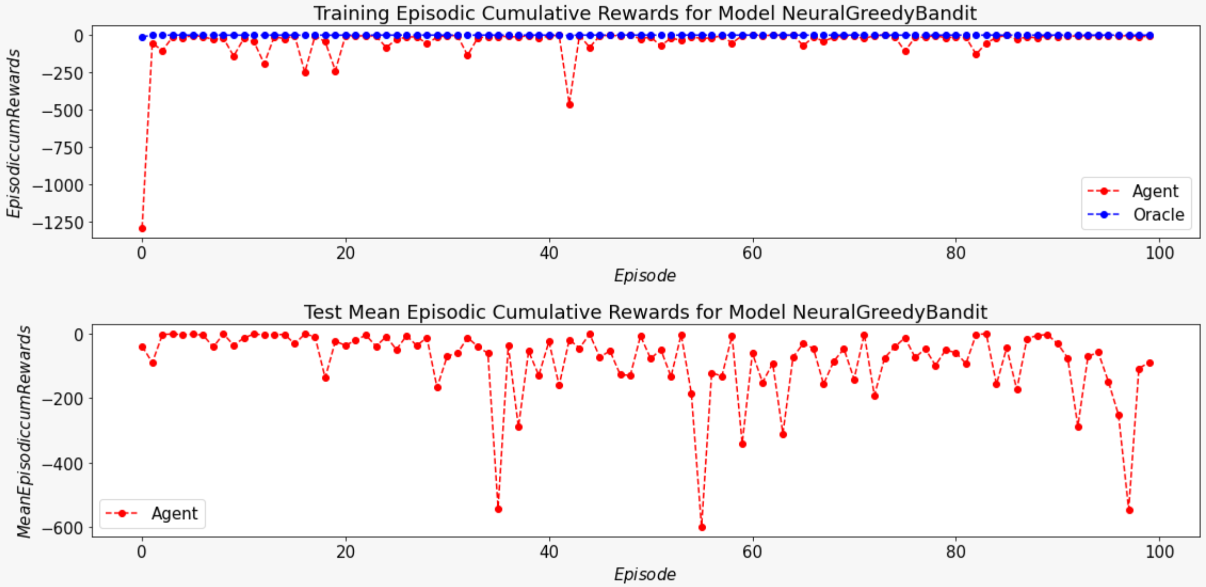
*Test performance based on 10 episodes of unseen data

Training Process and Test Performance

Neural Greedy Bandit

w. short-selling

w.o. short-selling

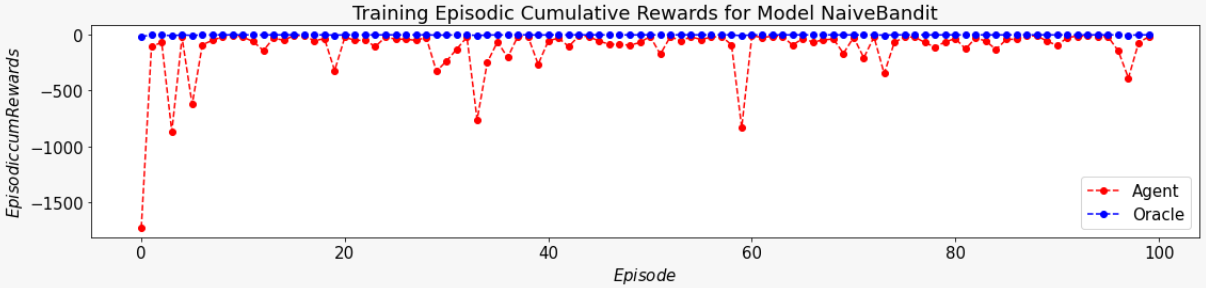


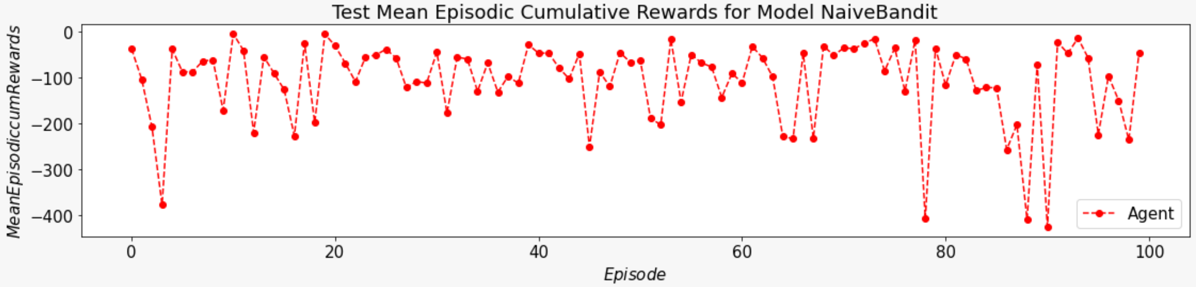
*Test performance based on 10 episodes of unseen data

Training Process and Test Performance

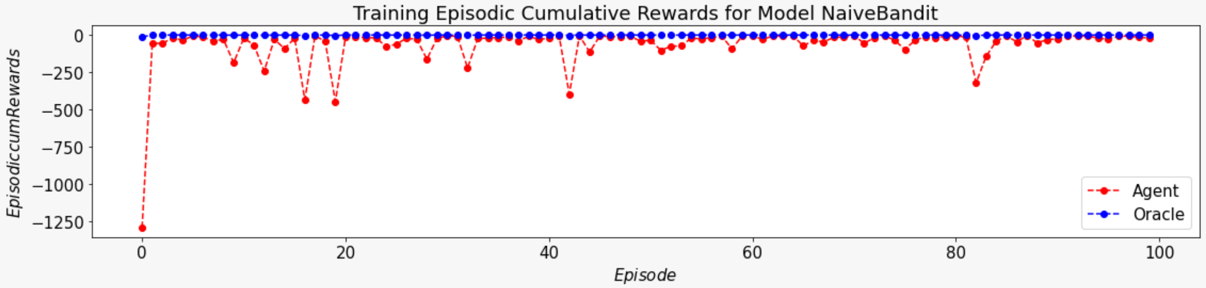
Naïve Bandit

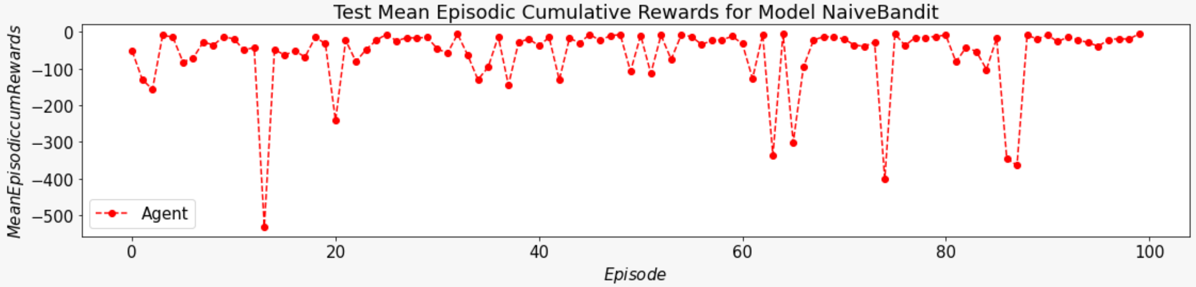
w. short-selling





w.o. short-selling



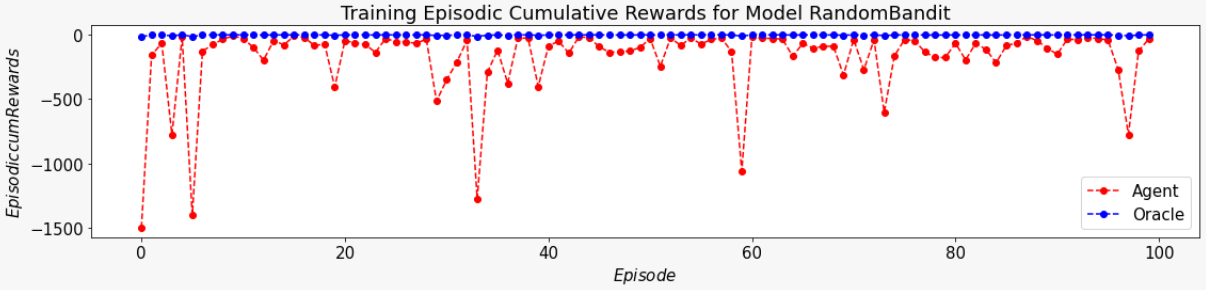


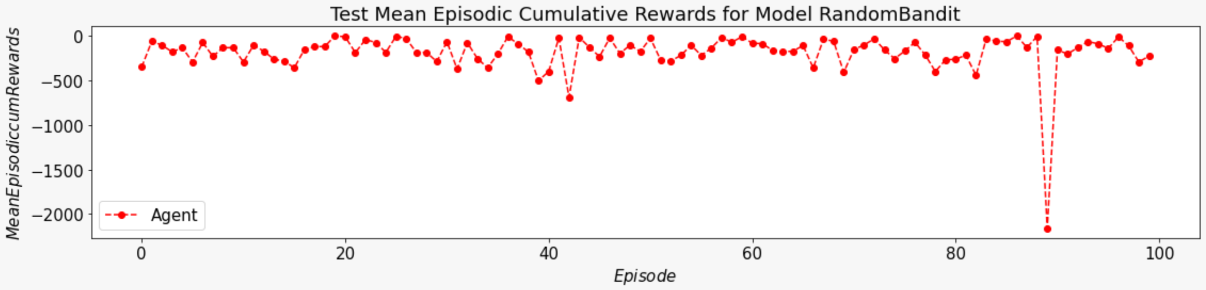
*Test performance based on 10 episodes of unseen data

Training Process and Test Performance

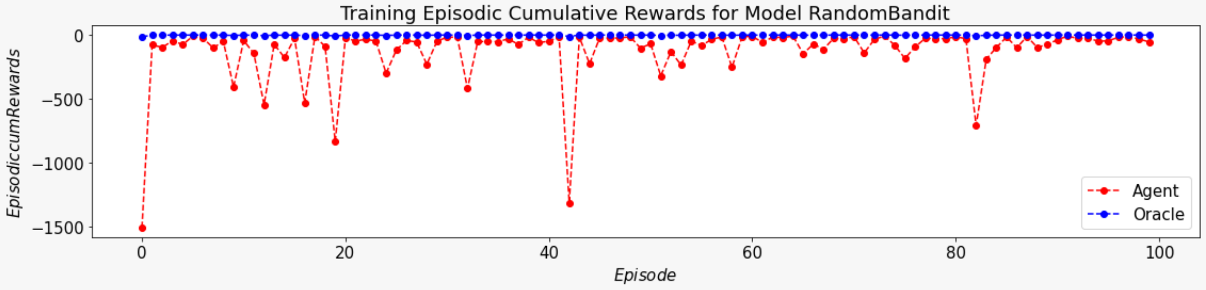
Random Bandit

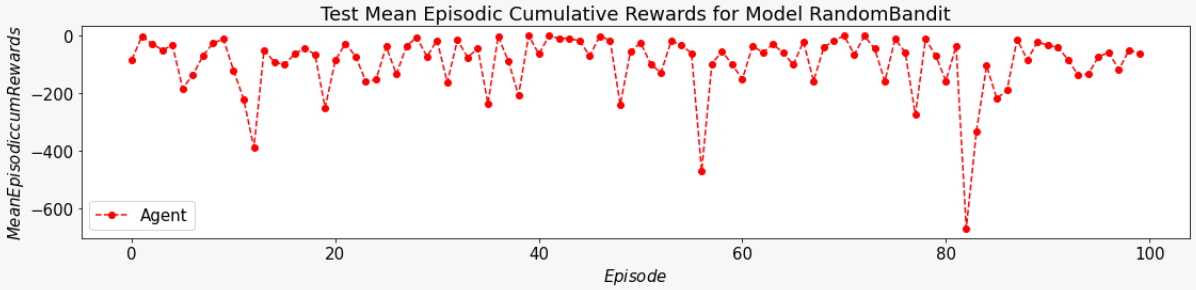
w. short-selling





w.o. short-selling



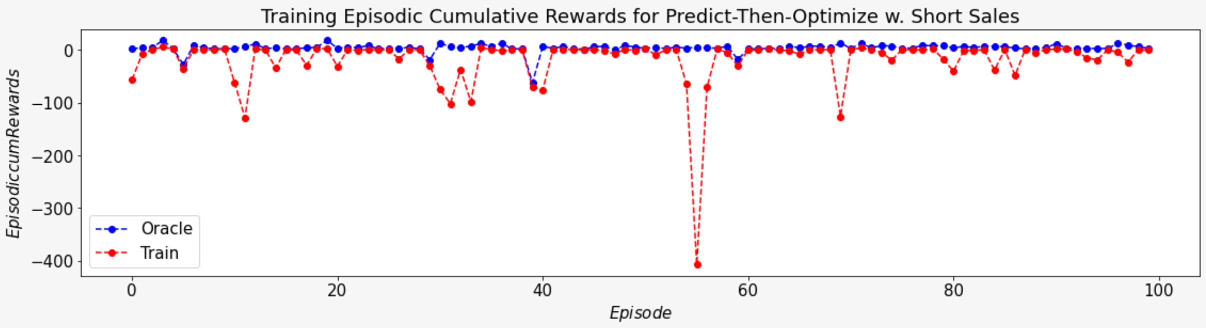


*Test performance based on 10 episodes of unseen data

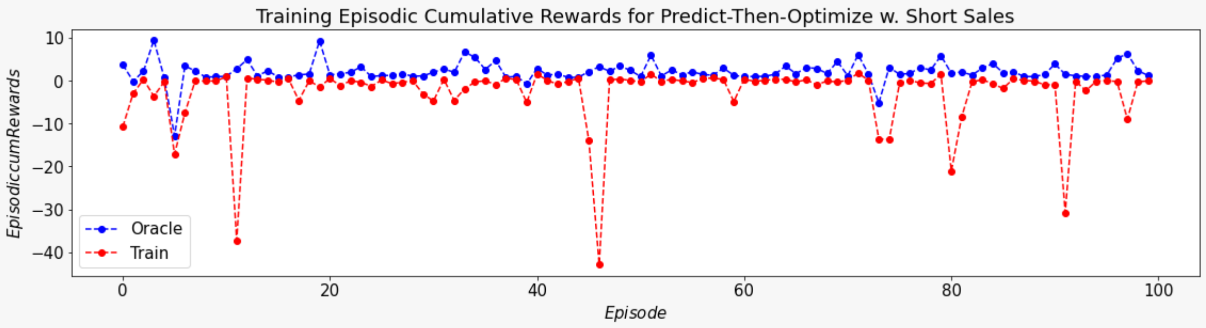
Training Process and Test Performance

SARIMAX

w. short-selling



w.o. short-selling

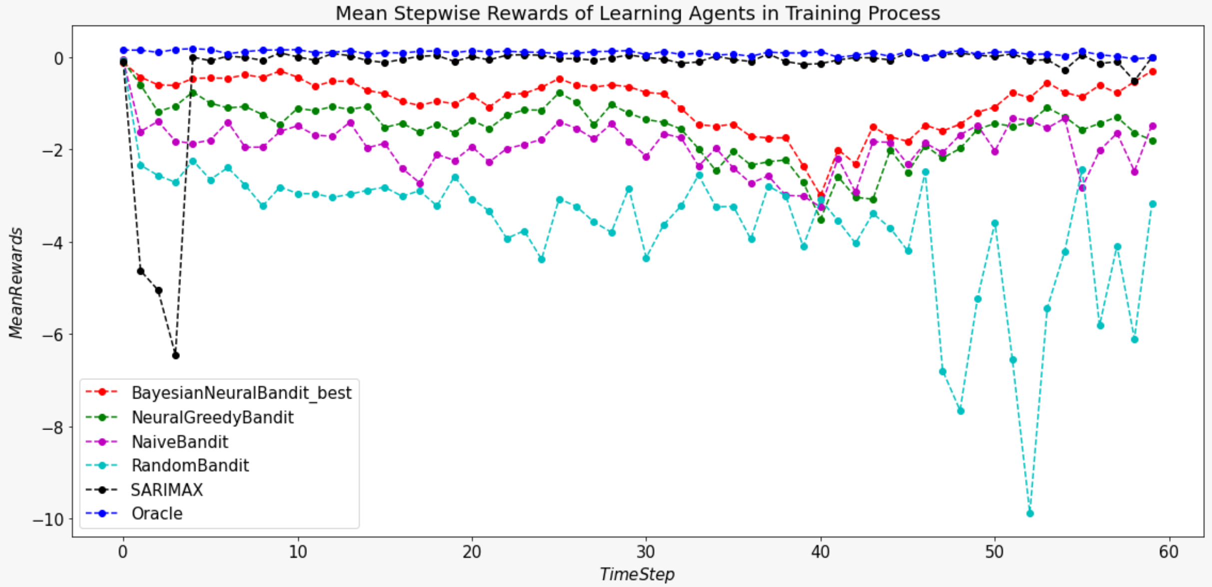


*Test performance based on 10 episodes of unseen data

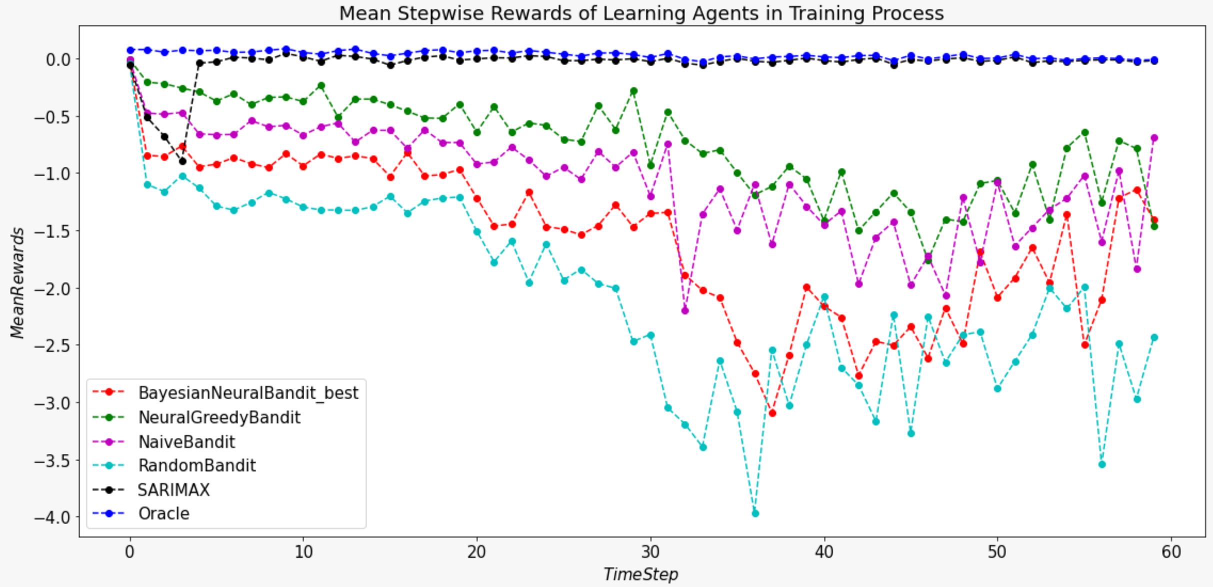
Training Process and Mean Stepwise Rewards

All Learning Agents

w. short-selling



w.o. short-selling

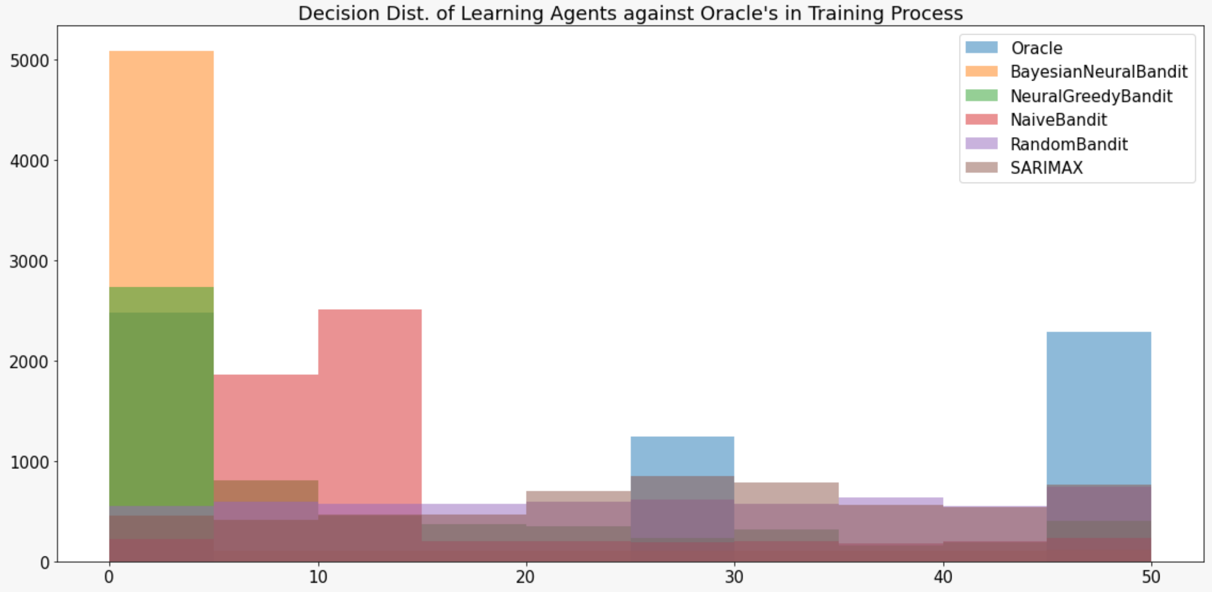


*Mean stepwise rewards averaged over training 100 episodes

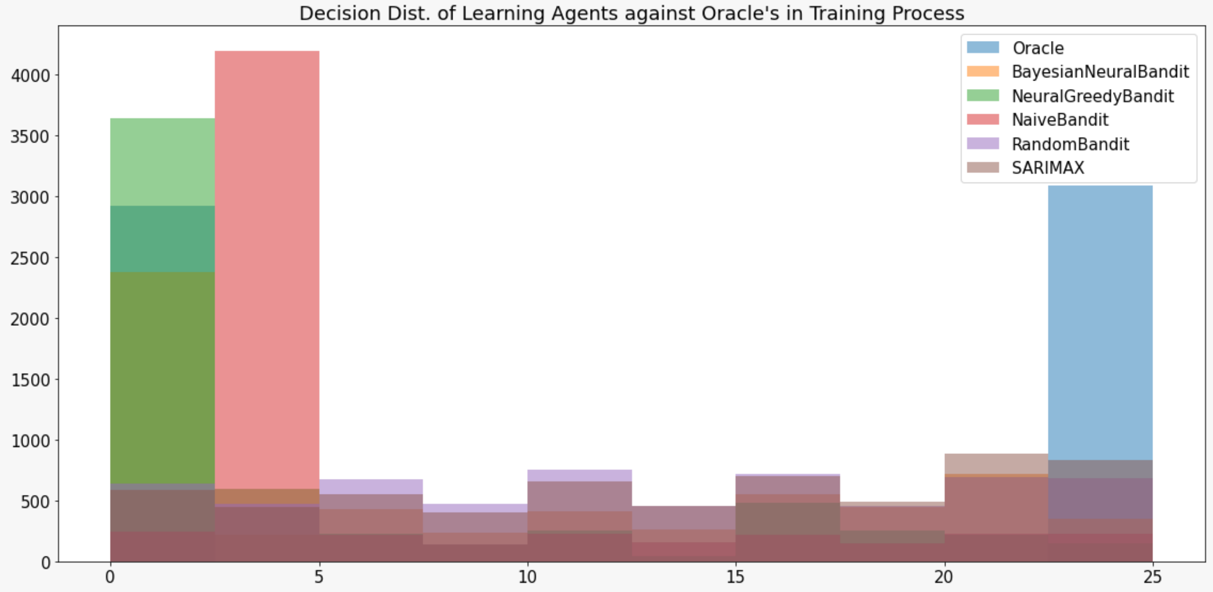
Training Process and Decision Distributions

All Learning Agents

w. short-selling



w.o. short-selling

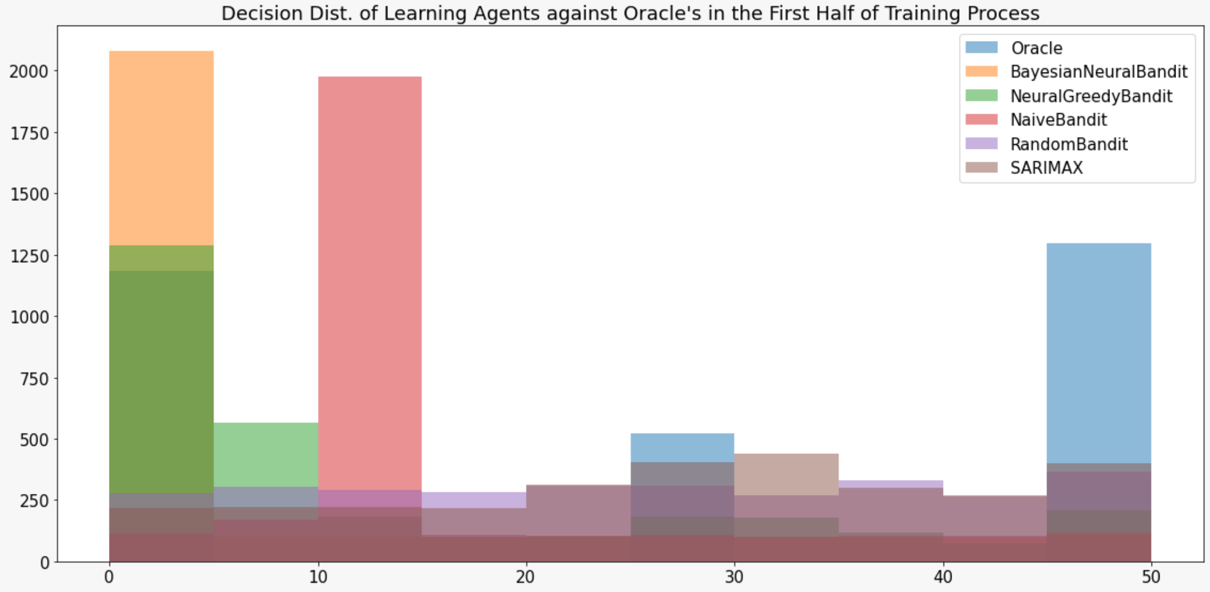


*Mean stepwise rewards averaged over training 100 episodes

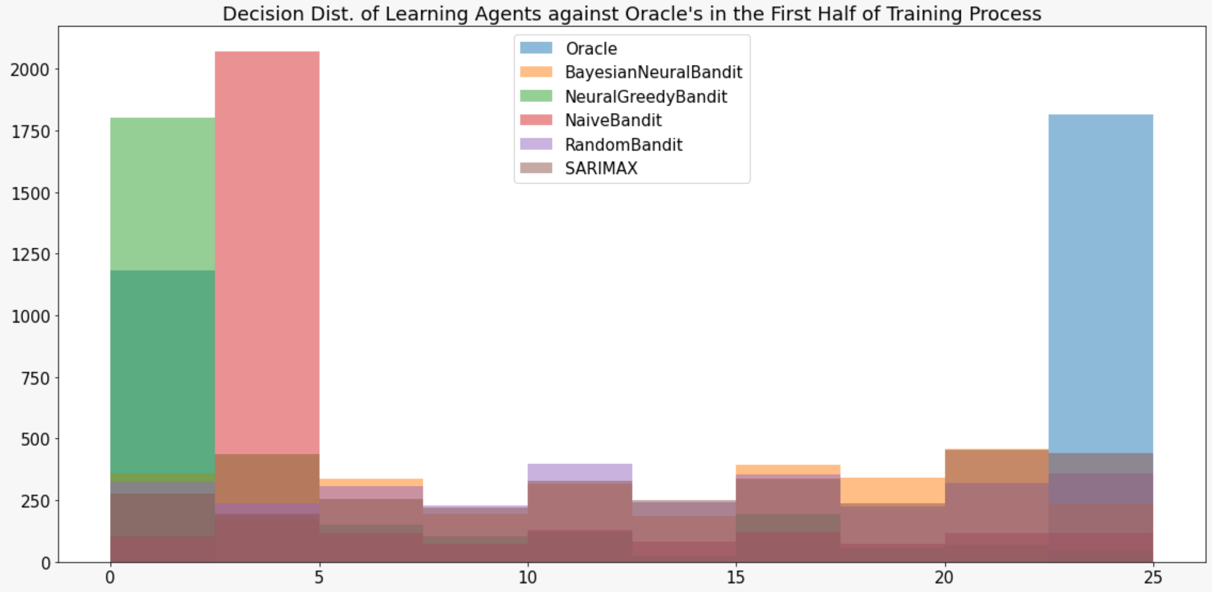
Training Process and Decision Distributions 1st Half

All Learning Agents

w. short-selling



w.o. short-selling

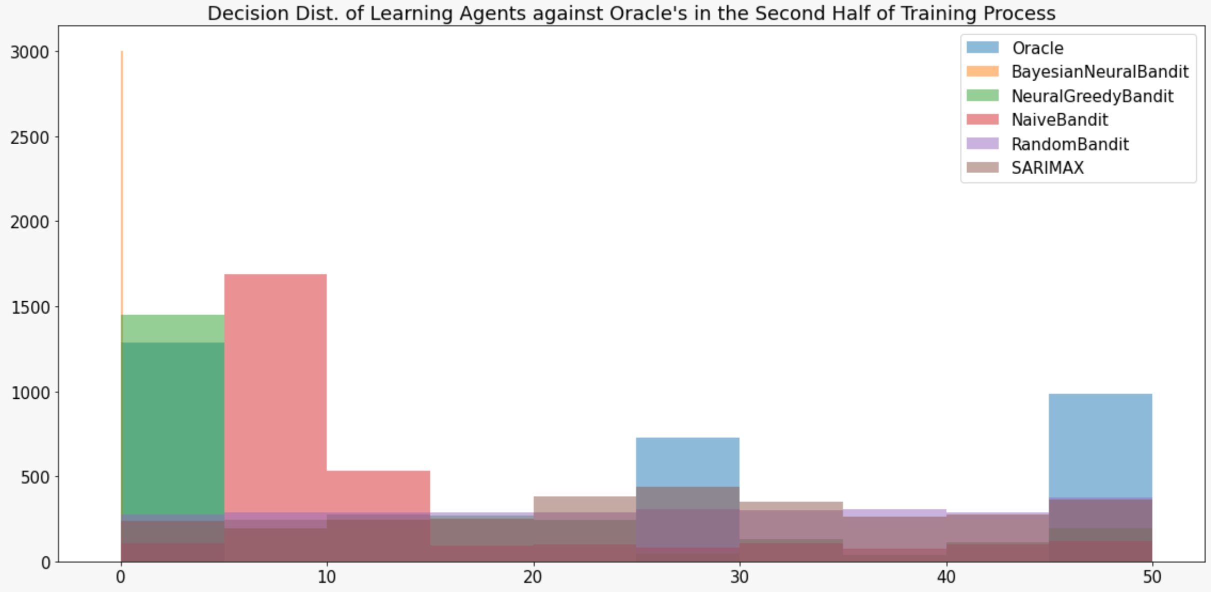


*Mean stepwise rewards averaged over training 100 episodes

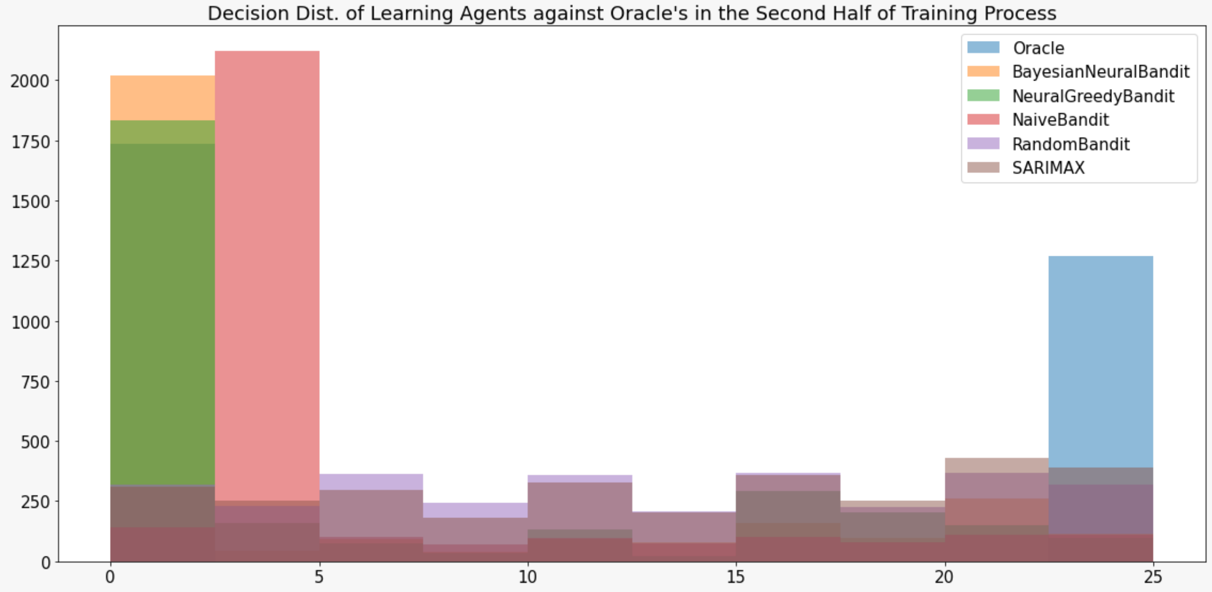
Training Process and Decision Distributions 2nd Half

All Learning Agents

w. short-selling



w.o. short-selling



*Mean stepwise rewards averaged over training 100 episodes

04

Conclusion

Key Findings and Limitations

Key Findings

- Replication strategies (both w./w.o. short-selling) in the presence of risk and market friction fall within the modeling capabilities of Contextual Multi-Armed Bandits
- Deep Neural Networks representation learning and Bayesian Linear Regression with Thompson Sampling technique can significantly improve bandit performance, balancing exploration and exploitation, but are sensitive to outliers and not desirably robust w.r.t. hyperparameters
- Times Series models do well in modeling the variance in market simulation, successfully mitigating losses in face of outliers, and are more robust in terms of hyperparameter settings
- Approximation of objective function greatly varies model convergence
- Bandits typically pay great attention to time-to-maturity and converge to Oracle’s behavior as the hedging process comes to the end
- Bandits are prone to short-selling to maximize rewards when stock/option prices are declining, while largely missing out the opportunities to hedge as stock/option prices go up

Limitations

- Our agents are trained and tested on simulated data generated from standard methods, the simulation strategy can have great impacts on model performance
- Some approximations of the objective function may not have guaranteed global convergence
- Agents warrant further hyperparams tuning, given limited time and computation resources
- It’s worthy of continuing research and more experiments to fully investigate the scope of the agents’ modeling capabilities and reasons behind variant agent behaviors under different formulations of the problem and combinations of hyperparameters

Contributions

LI, WENHAO

50%

SUN, ZHITING

50%

References

- Buehler, H., Gonon, L., Teichmann, J., Wood, B., Mohan, B., & Kochems, J. (2019). Deep hedging: hedging derivatives under generic market frictions using reinforcement learning. Swiss Finance Institute.
- Cannelli, L., Nuti, G., Sala, M., & Szehr, O. (2020). Hedging using reinforcement learning: Contextual k-Armed Bandit versus Q-learning. arXiv:2007.01623.
- Cao, J., Chen, J., Hull, J., & Poulos, Z. (2021). Deep hedging of derivatives using reinforcement learning. *The Journal of Financial Data Science, 3*(1), 10-27.
- Kolm, P. N., & Ritter, G. (2019) Dynamic replication and hedging: A reinforcement learning approach. *The Journal of Financial Data Science, 1*(1), 159–171.
- Riquelme, C., Tucker, G., & Snoek, J. (2018). Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. arXiv:1802.09127.
- Russo, D. J., Van Roy, B., Kazerouno, A., Osband, I., & Zheng, W. (2017). A tutorial on Thompson Sampling. arXiv:1707.02038
- Sattar, V., & Qing, Z. (2016). Risk-averse multi-armed bandit problems under mean-variance measure. *IEEE Journal of Selected Topics in Signal Processing, 10*(6), 1093–1111.
- Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.