Mila

RandUCB Optimization

Yutong Yan

McGill
School of Computer Science

Outline

- **Randomized upper confidence bound (RandUCB)**
- **Projection onto simplex**
- **Bandit instances settings**
- **Deep learning approach**
- **Evolution strategy approach**
- **Results**
- **Future work**

Multi-armed bandit

Algorithm 1: MAB algorithm

Input: Action set $\mathcal{A}$

for *time step* $t = 1, 2, 3, \ldots$ **do**

  Select i_t from $\mathcal{A}$

  Get reward $r_t \sim \mathcal{D}_{i_t}$

  Update i_t using r_t

end

- **Maximize** $\sum_{t=1}^{T} r_t$
- **By choosing optimal action:**

$$i_\star = \underset{i \in \mathcal{A}}{\operatorname{argmax}}\, \mu_i \quad \text{, where}$$

$$\mu_i = \mathbb{E}[\mathcal{D}_i]$$

- **Minimize expected regret:**

$$R(T) = \sum_{t=1}^{T} [\mu_\star - \mu_{i_t}]$$

Randomized upper confidence bound (RandUCB)

- **Upper Confidence Bound (UCB)**
  - **The policy to select the arm is**

$$i_t = \underset{i \in \mathcal{A}}{\operatorname{argmax}} \{\hat{\mu}_i(t) + \beta \mathcal{C}_i(t)\}$$

- **RandUCB**
  - **The policy to select the arm is**

$$i_t = \underset{i \in \mathcal{A}}{\operatorname{argmax}} \{\hat{\mu}_i(t) + Z_t \mathcal{C}_i(t)\}$$

The sampling distribution

- **Consider a discrete distribution on [*L*, *U*]**
- **There are *M* points in the distribution**

$$\alpha_1 = L, \ldots, \alpha_M = U$$

$$p_m := P(Z = \alpha_m)$$

- **UCB algorithm: M=1, L=U=β**
- **Optimistic: L=0**
- **Non-optimistic: L=-U**

Motivation of this work

- RandUCB uses a truncated Gaussian distribution [1]
  - Works fine for MAB, Linear bandits (LB)
  - Fails at tree search bandits
- The goal of this work
  - Finds a discrete probability distribution
  - Outperforms Gaussian distribution
  - Works in all bandit variants

Projection Method

- **We implement the projection method proposed in [2]**

Algorithm 2: Projection [2]

Input: $\mathbf{y} \in \mathbb{R}^D$

Sort $\mathbf{y}$ into $\mathbf{u}$: $u_1 \geq u_2 \geq ... \geq u_D$

Find $\rho = \max\{1 \leq j \leq D : \mu_j + \frac{1}{j}(1 - \sum_{i=1}^{j} \mu_i) > 0\}$

Define $\lambda = \frac{1}{\rho}(1 - \sum_{i=1}^{\rho} \mu_i)$

Output: $\mathbf{x}$ s.t. $x_i = \max\{y_i + \lambda, 0\}, i = 1, ..., D.$

Bandit Instances Setting

- **Bandit class**
  - **Easy class: means uniformly sampled from [0.25, 0.75]**
  - **Hard class: means uniformly sampled from [0.45, 0.55]**
- **Oracle**
  - **Takes a discrete probability distribution (probability vector) as input**
  - **Outputs the regret measured on a group of bandit instances**

$$R_T(p)$$

Deep Learning Approach

- **Takes a probability vector as input**
- **Outputs the estimated regret**

$$\mathrm{NN} : \mathbf{p} \rightarrow \hat{R}_T(\mathbf{p})$$

- **Data Generation:**
  - **Random vectors after projection**
  - **Normalized random vectors**
  - **Disturbed Gaussian vectors**

Neural Network Model

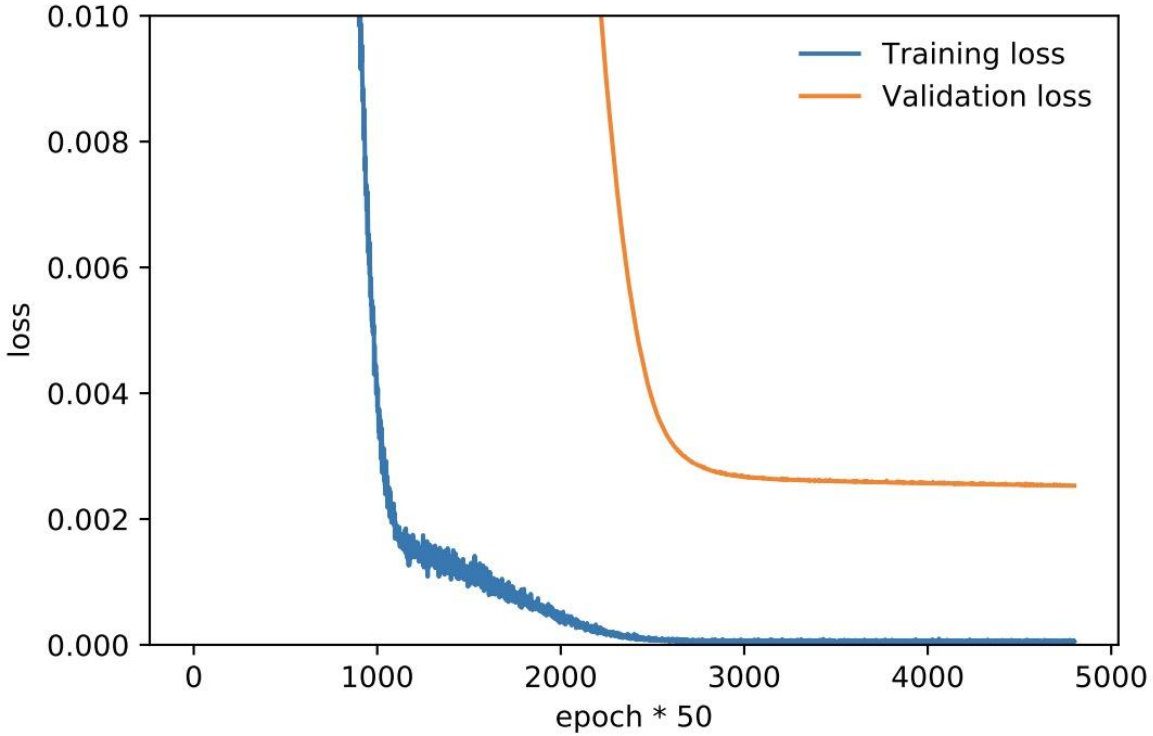
- **Only one layer to avoid overfitting**

Table 1: Neural Network Model Details

Input Size	Hidden Layer Size	Output Size	Activation Function	Learning Rate
10	5	1	Sigmoid	1e-3

Loss Function	Optimizer	Weight Decay	Epochs	Batch Size
MSE	Adam	1e-5	100	50

Train & Valid Loss



RandUCB Optimization Using Neural Networks

Algorithm 3: RandUCB Optimization Using NN

Input: learning rate lr
Initialize $\mathbf{p_0}$ randomly
$\hat{\mathbf{p}}_\mathbf{0} \leftarrow \mathbf{p_0}$
while *convergence condition not met* **do**
  $\hat{R}_T(\mathbf{p}_0) \leftarrow \text{NN}(\mathbf{p_0})$
  ORACLE outputs $R_T(\mathbf{p}_0)$
  if $R_T(\mathbf{p}_0) < R_T(\hat{\mathbf{p}}_\mathbf{0})$ **then**
    $\hat{\mathbf{p}}_\mathbf{0} \leftarrow \mathbf{p_0}$
  end
  Loss $\leftarrow$ MSE($\hat{R}_T(\mathbf{p}_0)$ - $R_T(\mathbf{p}_0)$)
  NN computes $\nabla \mathbf{p_0}$ w.r.t Loss
  $\mathbf{p_0} \leftarrow \mathbf{p_0}$ - lr * $\nabla \mathbf{p_0}$
  $\mathbf{p_0} \leftarrow$ project($\mathbf{p_0}$)
end
Output: $\hat{\mathbf{p}}_\mathbf{0}$

- **Autograd computes the gradient of input using requires_grad=True**
- **Convergence condition:**
  - **The grad vector becomes zero vector**
    - **Less than 10 iterations**
  - **Meets max iterations**
    - **Usually 25 - 50**
- **Learning rates**
  - **0.01, 0.0075, 0.005, 0.0025**
  - **Not very sensitive on the results if within a range**
- **Random Initialization**
  - **50 for each learning rate**

Evolution Strategy Approach

- **Takes a probability vector as input**
- **Outputs the regret**

$$f : \mathbf{p} \rightarrow R_T(\mathbf{p})$$

- **We use Python tool pycma with implemented CMA-ES**

Evolution Strategy Approach

Algorithm 4: RandUCB Optimization Using ES

Input: initial probability vector $\mathbf{p}_0$, sigma σ
es $\leftarrow$ CMAES($\mathbf{p}_0$, σ)
while *stopping criteria not met* **do**
    $\hat{\mathbf{p}} \leftarrow$ es.ask()
    ORACLE outputs $R_T(\text{project}(\hat{\mathbf{p}}))$
    es.tell($\hat{\mathbf{p}}$) $\leftarrow R_T(\text{project}(\hat{\mathbf{p}}))$
end
$\hat{\mathbf{p}} \leftarrow$ es.fmin_x
$\hat{\mathbf{p}} \leftarrow$ project($\hat{\mathbf{p}}$)
Output: $\hat{\mathbf{p}}$

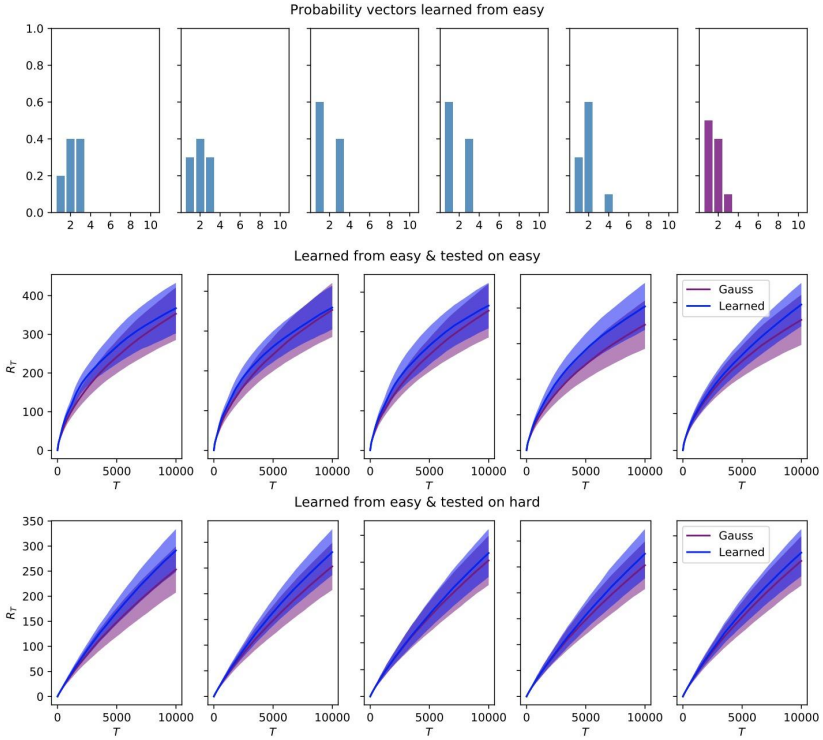
Comparison

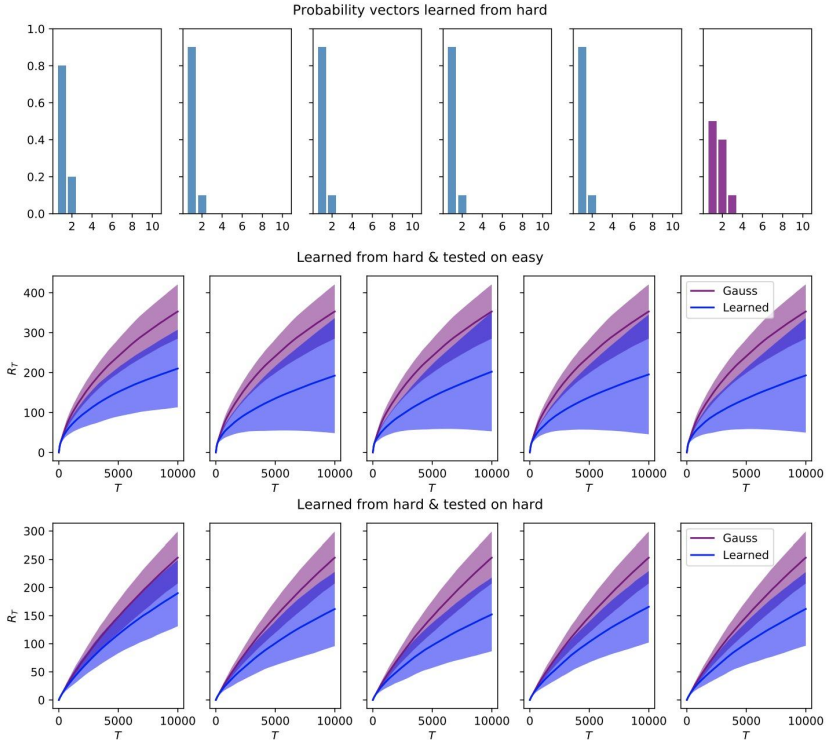
- **Evolution Strategy**
  - **Derivative-free**
  - **No data collection needed**
  - **No optimization step needed**
- **Deep Learning**
  - **Guarantees a solution (might not be the best)**
  - **Takes less time to obtain a result (excluding data collection)**

Experiments

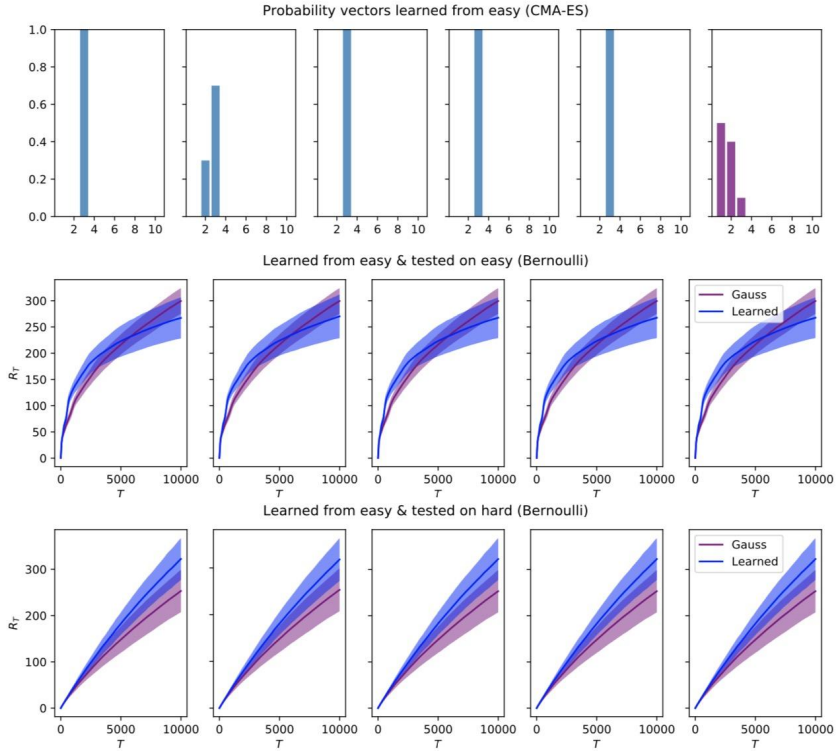
- **Single Bandit Class (Easy and Hard)**
- **Hybrid Bandit Class**
- **Non-Optimistic**
- **Size of probability vector M=2**
- **Comparison between more (1000) and less (10) bandit instances**
- **Performance on linear bandits**

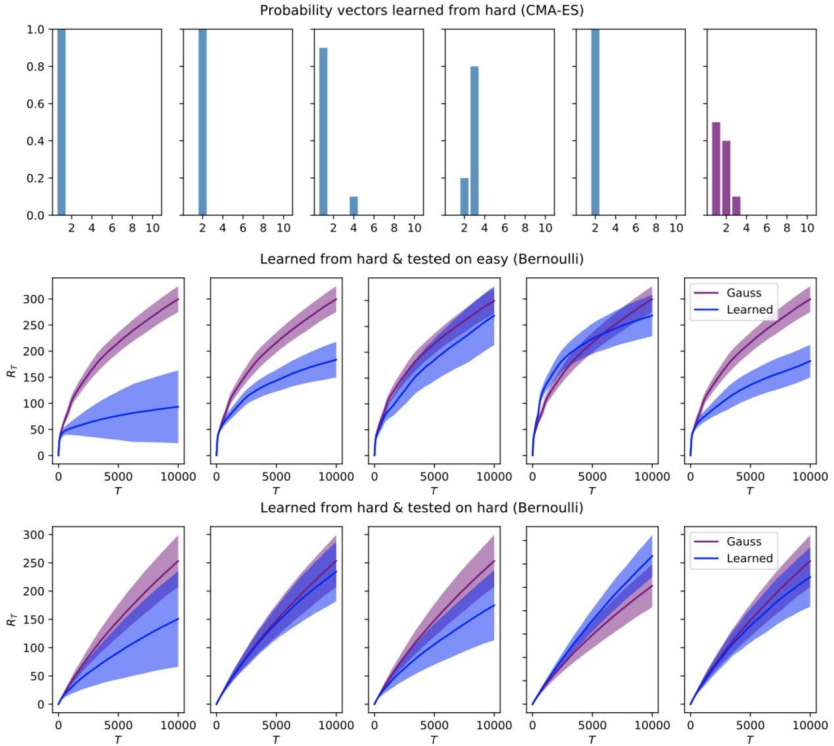
Single Bandit Class (Deep Learning)





Single Bandit Class (Evolution Strategy)

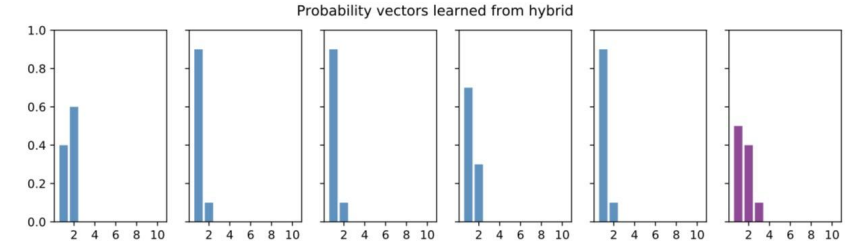


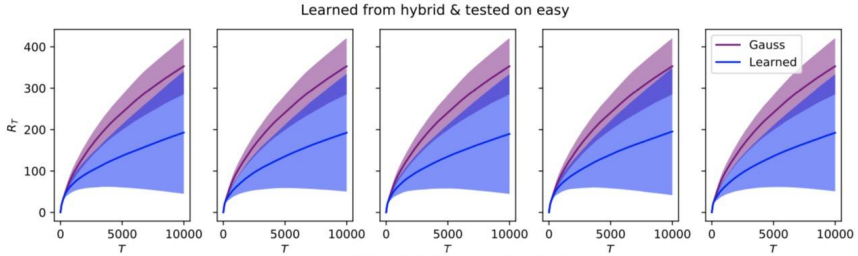


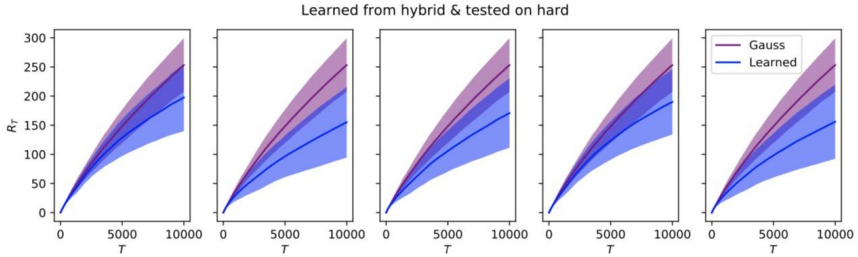
Disadvantages of Evolution Strategy

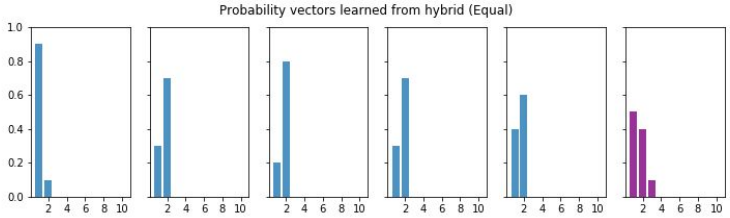
- Cannot always provide a solution when the algorithm stops
- Too deterministic
- When the problem is more complex (hard), the performance is not consistent

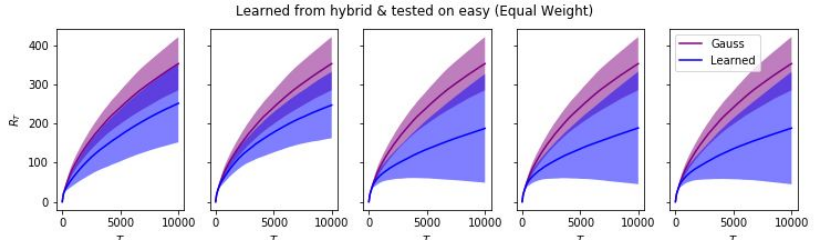
Hybrid Bandit Class (Deep Learning)

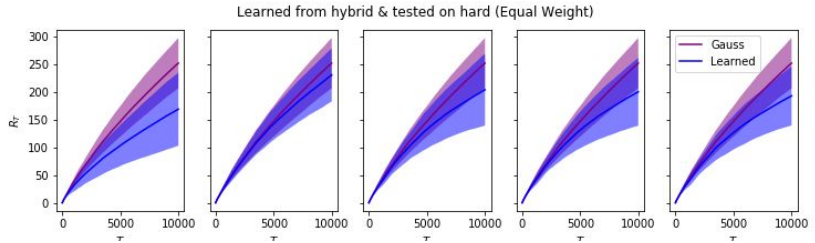




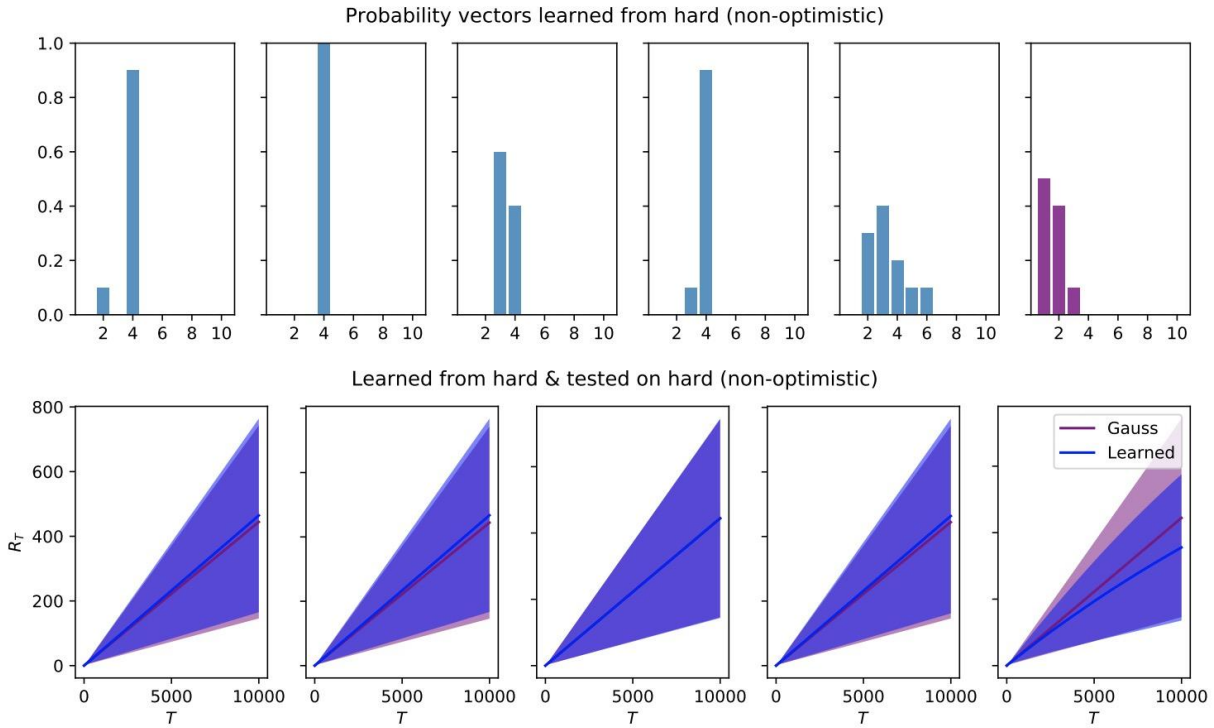




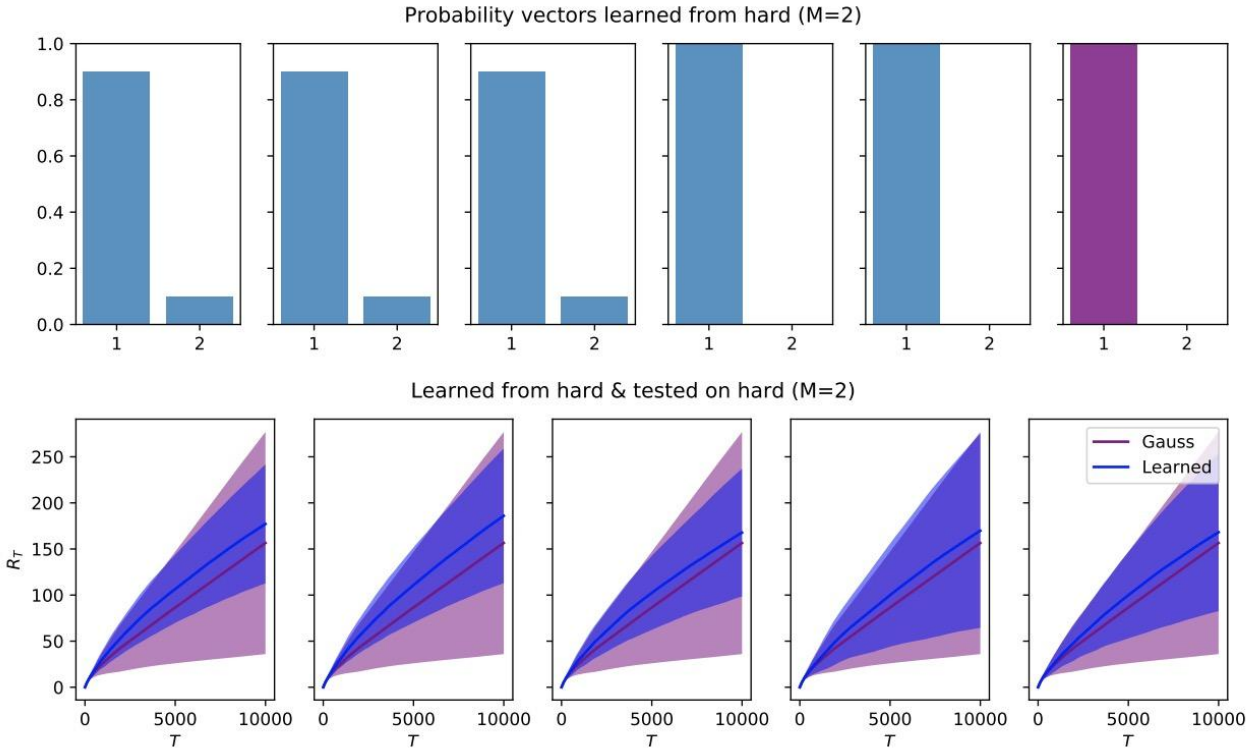




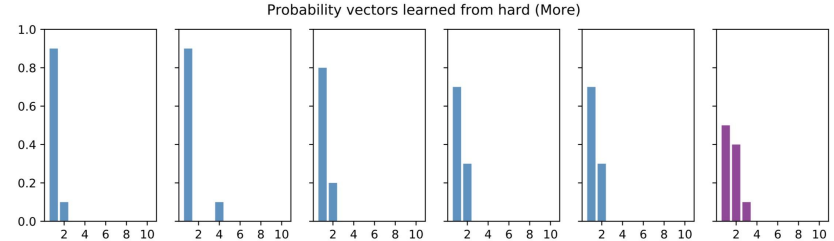
Non-Optimistic Case

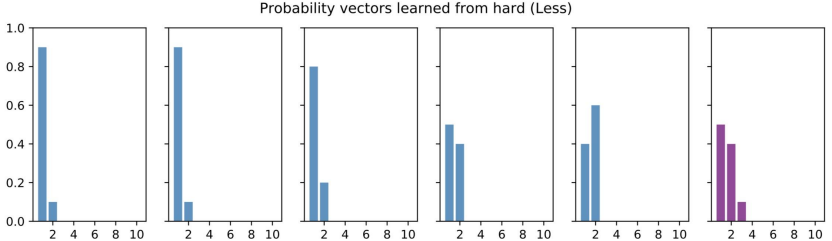


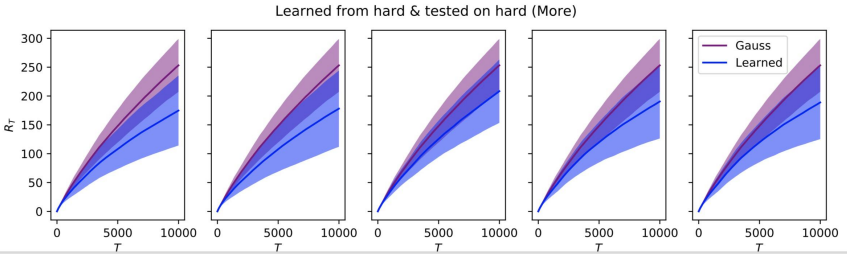
Size of probability vector M=2

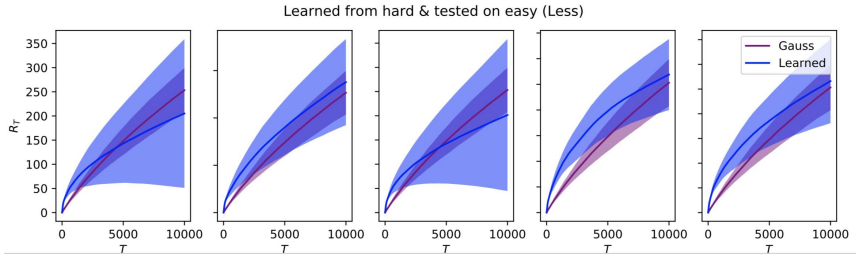


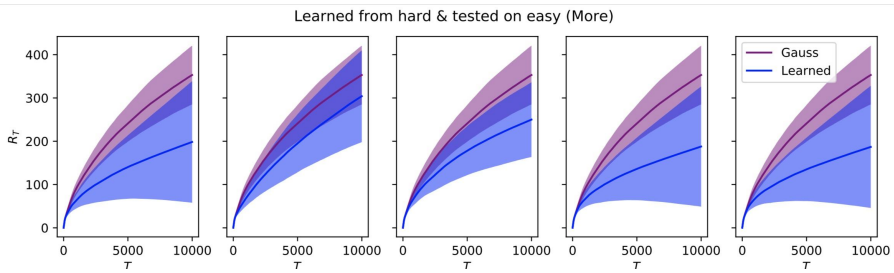
Comparison between more (1000) and less (10) bandit instances

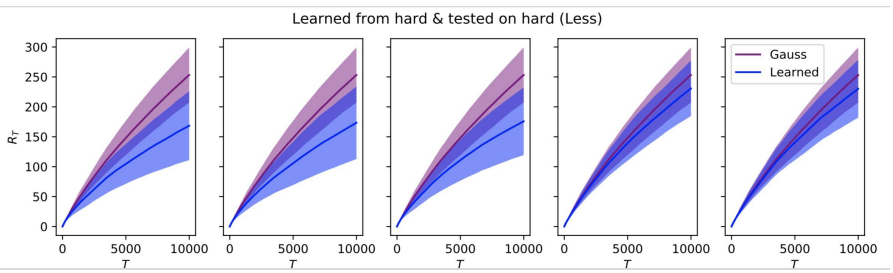




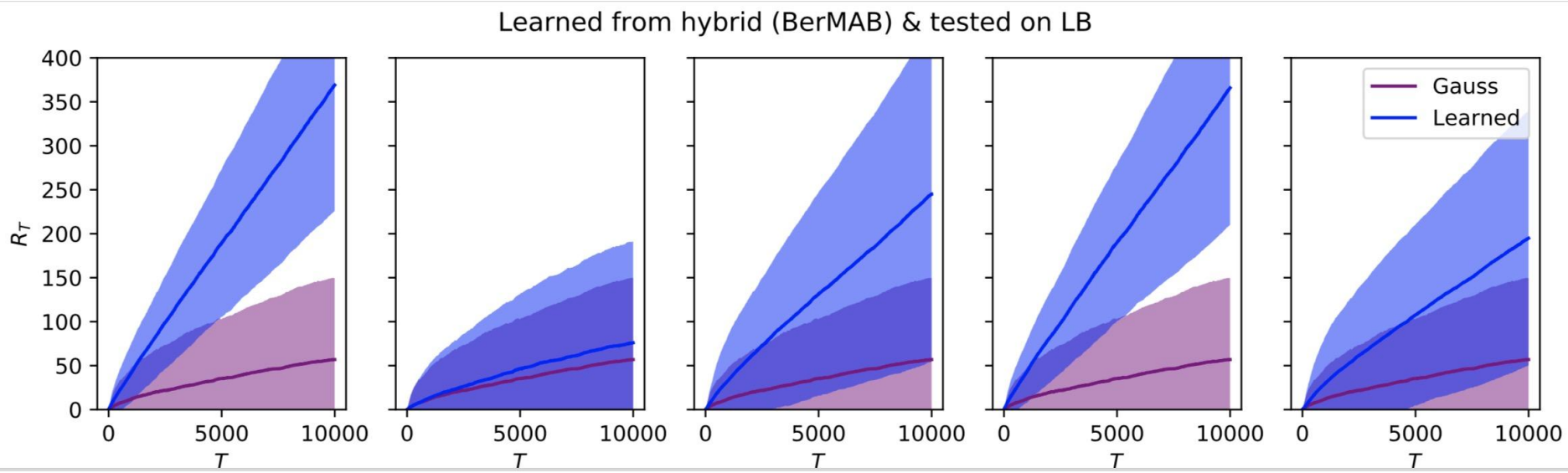








Performance on linear bandits



Future Work

- Collect data using 1000 bandit instances
- Apply deep learning approach on linear bandits and weighted linear bandits [3]
- If the results are promising, propose a randomized tree search algorithm

Future Work

[1] Sharan Vaswani, Abbas Mehrabian, Audrey Durand, and Branislav Kveton. Old dog learns new tricks: Randomized ucb for bandit problems. arXiv preprint arXiv:1910.04928, 2019.

[2] Weiran Wang and Miguel A Carreira-Perpinan. Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application. arXiv preprint arXiv:1309.1541, 2013.

[3] Yoan Russac, Claire Vernade, and Olivier Cappe. Weighted linear bandits for non-stationary environments. In Advances in Neural Information Processing Systems, pages 12017–12026, 2019.