A Simple Dynamic Bandit Algorithm for Hyper-parameter Tuning

Xuedong Shang\textsuperscript{1,2}, Emilie Kaufmann\textsuperscript{1,2,3}, Michal Valko\textsuperscript{4,1}

\textsuperscript{1}Inria Sequel \textsuperscript{2}Univ. Lille \textsuperscript{3}CNRS \textsuperscript{4}DeepMind Paris

Problem and Objectives

We treat the hyper-parameter tuning problem for supervised learning tasks.

- global optimisation task: \( \min \{ f(\lambda) : \lambda \in \Omega \} \);
- \( f(\lambda) \triangleq \mathbb{E} \{ Y, \mathcal{g}_\lambda(X) \} \) measures the generalization power.

Our contribution: a simple, robust, (almost) parameter-free bandit algorithm.

How and Why

How?

We see the problem as best arm identification in a stochastic infinitely-armed bandit: arms' means are drawn from some reservoir distribution \( \nu_0 \).

\( \nu_0 \) is observed, the algorithm is updated with a fake binary reward \( Y' \sim \text{Ber}(Y) \) and run TTTS on the new set of arms.

Why?

- \texttt{TTTS} is anytime for finitely-armed bandits
- \texttt{TTTS} is dynamic for the flexibility of this Bayesian algorithm allows to propose a dynamic version for the infinite BAI
- \texttt{TTTS} does not need to fix the number of arms queried in advance, and naturally adapts to the difficulty of the task

D-TTTS \( \Rightarrow \) a dynamic algorithm built on TTTS [1]

In the Context of BAI...

- **Beta-Bernoulli** Bayesian bandit model
  - a uniform prior over the mean of new arms
  - Posterior distribution on arm \( i \) at time \( t \):
    \[ \text{Beta}(1 + S_i, N_i - S_i + 1). \]

D-TTTS principle: in each round, query a new arm endowed with a \( \text{Beta}(1,1) \) prior, without sampling it, and run TTTS on the new set of arms.

Implementation tricks

**Binarization trick:** When a reward \( Y_{ij} \in [0,1] \) is observed, the algorithm is updated with a fake binary reward \( Y' \sim \text{Ber}(Y) \).

**Order statistic trick:** with \( \mathcal{L}_{t-1} \), the list of arms that have been effectively sampled at time \( t \), we run TTTS on the set \( \mathcal{L}_{t-1} \cup \{ \mu_0 \} \) where \( \mu_0 \) is a pseudo-arm with posterior \( \text{Beta}(t-|\mathcal{L}_{t-1}|,1) \).

Experimental Setting

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Hyper-parameter Type</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>( C )</td>
<td>( \mathbb{R}^+ )</td>
</tr>
<tr>
<td>SVM</td>
<td>( \gamma )</td>
<td>( \mathbb{R}^+ )</td>
</tr>
<tr>
<td>MLP</td>
<td>hidden_layer_size</td>
<td>Integer</td>
</tr>
<tr>
<td>MLP</td>
<td>alpha</td>
<td>( \mathbb{R}^+ )</td>
</tr>
<tr>
<td>MLP</td>
<td>learning_rate_init</td>
<td>( \mathbb{R}^+ )</td>
</tr>
</tbody>
</table>

Table: hyper-parameters to be tuned for UCI experiments.

Sampling Rule

1. **Input**: \( \beta \)
2. **Initialization**: \( \mu_1 \sim \nu_0; \mathcal{A} = \{ \mu_1 \}; S_i, N_i = 0 \)
3. **while** budget still available **do**
4. \( \mu_{m+1} \sim \nu_0; \mathcal{A} \leftarrow \mathcal{A} \cup \{ \mu_{m+1} \} \)
5. \( S_{m+1}, N_{m+1} \leftarrow 0; m \leftarrow m + 1 \)
6. \( \forall i \in \mathcal{A}, \theta_i \sim \text{Beta}(S_i + 1, N_i - S_i + 1) \)
7. \( I^{(1)} = \arg \max_{\theta \in \Theta} \theta_i \)
8. **if** \( U(\sim \mathcal{U}([0,1])) > \beta \) **then**
9. **while** \( I^{(2)} \neq I^{(1)} \) **do**
10. **forall** \( i \in \mathcal{A}, \theta_i' \sim \text{Beta}(S_i + 1, N_i - S_i + 1) \)
11. \( I^{(2)} \leftarrow \arg \max_{\theta' \in \Theta} \theta_i' \)
12. **end while**
13. **end if**
14. \( Y \leftarrow \text{evaluate arm } I^{(1)}, X \sim \text{Ber}(Y) \)
15. \( S_{p(1)} \leftarrow S_{p(1)} + X; N_{p(1)} \leftarrow N_{p(1)} + 1 \)
16. **end while**

Understanding the Algorithm

**Adaptation to the difficulty:** for a 'difficult' reservoir, the pseudo-arm \( \mu_0 \) is sampled more often (i.e. more arms are effectively sampled)

**Results for HPO**

- \( \Rightarrow \) efficiently sampled arms for \( \text{Beta}(1,1) \) reservoirs:
- \( \Rightarrow \) efficiently sampled arms for shifted \( \text{Beta} \) reservoirs:

References


Inria

DeepMind Paris