**λOpt: Learn to Regularize Recommender Models in Finer Levels**

Yihong Chen¹, Bei Chen², Xiangnan He², Chen Gao³, Yong Li³, Jian-Guang Lou⁴, Yue Wang⁵

¹ Tsinghua University, ² Microsoft Research, ³ University of Science and Technology of China

---

Matrix Factorization for Recommendation

![Diagram of Matrix Factorization](image)

**Figure 1:** Conventional MF with Bayesian Personalized Ranking (BPR) criterion.

### Regularization Tuning Headache

\[ l = l(\Theta) + \lambda |\Theta|^2 \]

Regularized Loss

- Bob
- Model

\[ l = l(\Theta) + \lambda |\Theta|^2 \]

Epoch

- AUC

- 0.85

- 0.60

- 0.45

**Figure 2:** The model can be highly sensitive to the choice of \( \lambda \).

Our Goal: Find the reasons behind the regularization tuning headache and design methods to automatically regularize recommender models within appropriate computation cost.

### Why Hard to Tune the Recommender Models?

**Hypothesis 1: Compromise on Regularization Granularity**

![User and Item Frequency Distributions](image)

1. Dataset: Long-tailed user and item frequencies!

2. Model: Different latent dimension counts differently!

**Figure 3:** Due to the characteristics of the models and datasets, fine-grained regularization often works better.

Typically, we use grid-search or babysitting to determine \( \lambda \). In such cases, we set a global \( \lambda \) instead of fine-grained \( \lambda \) as it would otherwise take unaffordable effort or computation cost.

**Hypothesis 2: Fixed Regularization Strength Throughout the Model Training Process**

![Diagram of Regularization Tuning](image)

**Figure 4:** Compared to the fixed approach (left), adaptive regularization (right) can enjoy more efficient exploration in \( \lambda \) space.

**How to Tackle the Regularization Tuning Problem for Recommender Models?**

Based on the above hypotheses, we propose λOpt to learn to regularize recommender models in finer levels.

### MF-BPR with Fine-grained Regularization

![Diagram of MF-BPR](image)

**Figure 5:** λOpt endows MF-BPR with fine-grained regularization.

**Alternating Optimization**

Regularization tuning can be regarded as a bi-level optimization problem

\[ \min_{\Theta} \sum_{(u,i,j) \in S_{\text{V}}} l(u,i,j) + \min_{\Lambda} \sum_{(u,i,j) \in S_{\text{S}}} l(u,i,j) \Theta, \Lambda, \]

At iteration \( t \):

- Fix \( \Lambda \), Optimize \( \Theta \) to almost the same as conventional MF-BPR except that \( \lambda \) is fine-grained
- Fix \( \Theta \), Optimize \( \Lambda \) to find a \( \Lambda \) which achieves the smallest validation loss \( \lambda \)

**Fix \( \Theta \), Optimize \( \Lambda \)**

Taking a greedy perspective, we look for a \( \Lambda \) which can minimize the next-step validation loss

- If we keep using current \( \Lambda \) for next step, we would obtain \( \Theta_{t+1} \)
- Given \( \Theta_{t+1} \), our aim is \( \min_{\Lambda} l_s(\Theta_{t+1}) \)

But how to obtain \( \Theta_{t+1} \) without influencing the normal \( \Theta \) update? Simulate the MF update!

- Obtain the gradients by combining the non-regularized part and penalty part

\[ \frac{\partial l_s}{\partial \Theta_{t+1}} = \frac{\partial l_s}{\partial \Theta_{t}} + \frac{\partial l_s}{\partial \Lambda} \]

Note that \( \Lambda \) is the only variable here.

- Simulate the operations that the MF optimizer would take

\[ \Theta_{t+1} = f(\Theta_{t}, \Lambda_{t}) \]

To avoid obscure derivation of gradients introduced by the MF optimizer and fine-grained regularization, we rely on auto-differentiation to implement λOpt. That is, we first prepare the non-regularized gradients using a simulate forward & backwards on training set. Then we use a \( \Lambda \)-network (where the weights is \( \Lambda \)) to do all the aforementioned gradient combination, MF optimizer simulation, and computation of validation loss. After a forward & backward pass of \( \Lambda \)-network, we get the \( \Lambda \) for next step.

**Results**

![Performance Comparison Table](image)

**Figure 6:** λOpt: fix \( \Theta \), update \( \Lambda \).

**Sparseness & Activeness**

![Sparseness & Activeness Diagram](image)

**Figure 7:** λOpt addresses both the sparse and active users.

**Analysis of \( \lambda \)-trajectory**

![λ-trajectory Diagram](image)

(a) For users on Amazon Review
(b) For items on Amazon Review

**Figure 8:** λOpt generates different \( \lambda \)-trajectories for different users/items.