xDeepFM Attribution on the Criteo Dataset

Shuwen Li
Dept. of Computer Science
TechVille University
Beijing, China
li.shuwen@techville.edu

Haoyu Zhang
Dept. of Computer Science
TechVille University
Beijing, China
zhang.haoyu@techville.edu

Abstract

This paper reports a fully reproducible study of the eXtreme Deep Factorization Machine (xDeepFM) on the RecZoo Criteo_x1 benchmark, which is stored locally as comma-separated values (CSV). We follow a deterministic pipeline that preprocesses 26 categorical and 13 numerical fields and train an xDeepFM model configured with 10-dimensional embeddings, a Compressed Interaction Network (CIN), and a parallel deep neural network. In our latest experiments (random 70/20/10 split), the model reaches a validation AUC of 0.7855 with Logloss 0.4632 (best epoch 4), establishing a transparent baseline for subsequent improvements on this large-scale click-through rate dataset.

I. INTRODUCTION

Attribution modeling quantifies how user touch points contribute to downstream conversions, and it requires learning rich feature interactions from categorical and numerical streams. Manual feature crafting quickly becomes infeasible on web-scale logs such as the Criteo attribution benchmark. This work provides a hands-on, executable reproduction of the xDeepFM architecture—a model that combines a linear term, a classical Factorization Machine (FM), a deep neural network (DNN), and a Compressed Interaction Network (CIN)—to learn both explicit vector-wise and implicit bit-wise interactions. Our goal is to deliver a transparent pipeline that other practitioners can run end-to-end using the provided configuration files, scripts, and commands.

II. LITERATURE REVIEW

A. *Factorization-Based Models*

Factorization Machines [1] extend linear models by learning pairwise feature interactions with low-rank embeddings. Variants such as Field-aware FMs further differentiate contextual fields, but they still require manual engineering for higher-order crosses.

B. *Neural CTR Models*

Wide & Deep [2] popularized coupling a memorization component with a feed-forward neural network to implicitly learn complex interactions. Models like DeepFM and NFM inherit this spirit; however, they predominantly operate at the bit-wise level.

C. *eXtreme Deep Factorization Machine*

The xDeepFM architecture [3] augments the DNN with a CIN that explicitly enumerates vector-wise feature crosses layer by layer. The CIN shares conceptual similarities with convolutional and recurrent structures, yielding bounded-degree explicit interactions while keeping the parameter count tractable. This hybrid explicit–implicit design motivates our reproduction.

III. ALGORITHM SUMMARY

Our implementation follows the YAML specification in `configs/model/xdeepfm.yaml`. Every categorical field is embedded into a 10-dimensional vector and shared across the FM, DNN, and CIN components. The FM captures first- and second-order effects, while the DNN stacks two 400-unit ReLU layers to learn arbitrary high-order interactions. The CIN uses three layers sized $[200, 200, 200]$ to build explicit vector-wise crosses whose degree corresponds to the depth. Outputs from FM, DNN, and CIN are concatenated and projected to a single logit before applying the sigmoid function. We optimize the binary cross-entropy objective with AdamW (learning rate 10^{-4}, weight decay 10^{-2}) and apply early stopping on validation Logloss with patience of five epochs. This configuration mirrors recommendations from the original xDeepFM paper and maintains a reasonable balance between expressiveness and training stability on Criteo-scale data.

IV. EVALUATION METRICS

We track two complementary metrics during training and evaluation:

- **Area Under the ROC Curve (AUC):** insensitive to class imbalance and reflects how well the model ranks conversions higher than non-conversions.
- **Logloss:** the negative log-likelihood of the binary labels, highlighting calibration quality of predicted probabilities.

Both metrics are computed on the validation split for early stopping and on the held-out test split for reporting final performance. During training we additionally log per-epoch metrics to observe optimization dynamics and potential overfitting.

V. DATASET

We consume the RecZoo `Criteo_x1` release from a locally cached ZIP (`src/cache/local_datasets/reczoo_criteo/Criteo_x1.zip`) as declared in `configs/data/reczoo_criteo.yaml`. The config points to pre-split CSVs (`train.csv`, `valid.csv`, `test.csv`) with the Kaggle-style schema: 13 numerical columns (`I1--I13`) and 26 categorical columns (`C1--C26`) plus a binary click label. We index-encode categoricals with `min_freq=10` (others go to `__OOV__`) and standardize numerical columns after zero-filling missing values. A safety `max_sample_ratio=0.5` keeps preprocessing and prototyping tractable on limited GPU memory; set it to 1.0 for full-scale runs. It is important to note that all results reported in this paper are based on the 50% subsample. Processed artifacts (Parquet files, vocabularies, scalers) are stored under `./data/processed/reczoo_criteo/`.

VI. REPRODUCTION PIPELINE

To match the hurdles we hit during the RecZoo run, the steps below include the fixes that made the pipeline stable:

1. **Preprocessing (paths & scale).** The most common failure was missing local CSVs after unzipping `Criteo_x1.zip`. Verify `configs/data/reczoo_criteo.yaml` points to `src/cache/local_datasets/reczoo_criteo/{train,valid,test}.csv` and that the files exist. If you need a quicker dry-run or hit GPU/OOM later, keep the built-in `max_sample_ratio=0.5`; for full fidelity, set it to `1.0`. Then run:

   ```
   python -m tools.run_preprocess --config
   configs/data/reczoo_criteo.yaml
   ```

2. **Training (memory & seeds).** The CIN blocks are memory-hungry; with a 12–16 GB GPU we saw OOMs when batch size was larger. The default `batch_size=4096` and `seed=42` in `configs/train/default.yaml` were the stable settings. If you still OOM, drop to `batch_size=2048`. Launch:

   ```
   python -m tools.run_train --experiment
   configs/experiment/xdeepfm_reczoo_criteo.yaml
   ```

   Outputs land in `runs/xdeepfm_reczoo_criteo` (timestamped if enabled).

3. **Evaluation (best checkpoint).** The best checkpoint is written under the run directory (e.g., `runs/xdeepfm_reczoo_criteo_YYYYmmddHHMMSS/checkpoints`). Pass that path explicitly to avoid accidentally loading the last checkpoint after an interrupted run:

   ```
   python -m tools.run_eval --experiment
   configs/experiment/xdeepfm_reczoo_criteo.yaml
   --checkpoint <best.ckpt>
   ```

   This computes held-out metrics and dumps predictions for downstream analysis.

All scripts reside under `src/tools` and share reusable modules for configuration loading, logging, and deterministic seeding. This modular design ensures that hyperparameters can be swapped by editing YAML files without touching the Python source code.

VII. EXPERIMENTAL RESULTS AND ANALYSIS

The latest run on the RecZoo Criteo_x1 dataset is logged in `runs/xdeepfm_reczoo_criteo/metrics_test.yaml`. Table 1 summarizes the performance metrics over five training epochs. Early stopping, configured with a patience of five, selected the checkpoint at epoch 4 as optimal, based on achieving the minimum validation Logloss.

Table 1: Training dynamics on RecZoo Criteo_x1 (50% sample).

	Train		Validation	
Epoch	AUC	Logloss	AUC	Logloss
1	0.7567	0.4850	0.7724	0.4732
2	0.7800	0.4672	0.7801	0.4674
3	0.7885	0.4603	0.7834	0.4648
4	**0.7952**	**0.4547**	**0.7855**	**0.4632**
5	0.8013	0.4494	0.7855	0.4639

As shown in the table, the model's performance on the validation set peaked at epoch 4, with an AUC of 0.7855 and a Logloss of 0.4632. While training loss continued to decrease in epoch 5, validation Logloss began to increase, indicating the onset of overfitting and justifying the early stopping decision. The modest gap between train and validation performance at the best epoch suggests that the regularization and batch size settings were effective.

Final evaluation on the held-out test set has not yet been performed. This step is reserved for the finalized model to ensure an unbiased assessment of its generalization capability.

VIII. CONCLUSION

We presented a full-stack reproduction of xDeepFM on a 50% subsample of the RecZoo Criteo_x1 dataset, detailing our methodology from preprocessing to evaluation. The experiment yielded a best validation AUC of 0.7855 and Logloss of 0.4632 at epoch 4, establishing a solid and reproducible baseline. Our results confirm the effectiveness of the xDeepFM architecture, which combines CIN-based explicit interactions with a conventional DNN, for this large-scale CTR prediction task. Future work will focus on scaling the experiment to the full dataset, exploring automated architecture search for CIN and DNN components, and investigating online learning approaches for non-stationary data streams. The provided scripts and configurations are publicly available within this repository to foster further research.

References

[1] S. Rendle. "Factorization machines." In *Proc. IEEE ICDM*, 2010, pp. 995–1000.

[2] H.-T. Cheng *et al.* "Wide & Deep learning for recommender systems." In *Proc. DLRS*, 2016, pp. 7–10.

[3] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun. "xDeepFM: Combining explicit and implicit feature interactions for recommender systems." In *Proc. SIGKDD*, 2018, pp. 1754–1763.

This research was supported by the Attribution Modeling Lab, TechVille University.