COURIER: Contrastive User Intention Reconstruction for Large-Scale Visual Recommendation : Appendix

1 Downstream usage of image embeddings

How the features are fed to the model is a critical factor that affects the performance of machine learning algorithms. For example, normalizing the inputs before input to neural networks is a well-known preprocessing that matters a lot. In our practice of pre-training and utilizing pre-trained embeddings in downstream tasks, we also find that the way we insert the pre-trained embeddings is critical to the downstream performance. We explore three different approaches: Vector, Similarity score, and Cluster ID.

Vector. The most straightforward and common practices of utilizing pre-trained embeddings are to use the embeddings as input to the downstream tasks directly. In such cases, the features of the item images are represented as embedding vectors, which is also the first approach we have tried. However, our experiments show that no matter how we train the image embeddings, the improvements in the CTR task are only marginal. We think the reason is that the embedding vectors are relatively hard to be used by the downstream model, so the downstream model ignores the embedding vectors. The existing recommender systems already use item IDs as features, and the embeddings of the IDs can be trained directly. So the IDs are much easier to identify an item than image embeddings, which require multiple layers to learn to extract related information. Thus, the model will achieve high performance by updating ID embeddings before the image-related weights can learn useful representations for CTR. Then the gradient vanishes because the loss is nearly converged, and the image-related weights won't update significantly anymore, which is also observed in our experiments.

Similarity Score. With the hypothesis about the embedding vectors, We experiment with a much-simplified representation of the embedding vectors. Specifically, assuming we want to estimate the CTR of an item with Img_{pv} and a user of click history Img^k_{click}. The vector approach uses Emb^k_{click} and Emb_{pv} as inputs directly. Instead, we calculate a cosine similarity score for

each click history items,

$$sim(Img_{pv}, Img^k_{click}) = \frac{Img^T_{pv} Img^k_{click}}{Img_{pv}\ Img^k_{click}} \quad (1)$$

and the scores (each image corresponds to a real-valued score) are used as image features. The results are in Table. (1). Experimental results indicate that the simple similarity score features perform significantly better than inserting embedding vectors directly, although the embedding vectors contain much more information. The performance of the similarity score method verified our hypothesis that embedding vectors may be too complex to be used in the CTR task.

Table 1 Performance of inserting image information with Vector, SimScore, and Cluster ID. Since we performed this comparison in the early stage of our development, the exact configurations of each version are hard to describe in detail. And the different versions may not be comparable to each other (different training data sizes, learning rates, training methods, etc.). We only list the version number for clarity. Results within each row are comparable since they are generated from the same version of embeddings. The Baseline does not use images. "-" denotes that we did not evaluate Cluster-ID of these versions.

	Vector	SimScore	Cluster ID
Baseline	0.00%	0.00%	0.00%
V1	0.07%	0.14%	0.23%
V2	0.08%	0.16%	0.23%
V3	0.09%	0.18%	0.28%
V4	0.06%	0.16%	0.22%
V5	0.00%	0.11%	-
V6	-0.02%	0.07%	-
V7	-0.04%	-0.01%	-
V8	0.04%	0.13%	-
V9	0.05%	0.09%	-

Cluster IDs. Our experiments on embedding vectors and similarity scores indicate that embedding vectors are hard to use and simply similarity scores contain information that can improve downstream performance. Can we insert more information than similarity scores but not too much? ID features are the most important features used in recommender systems, and it has been observed that ID features are easier to train than non-ID features, and perform better[1]. Thus, we propose to transform embedding vectors into ID features. A straightforward and efficient method is to hash the embedding vectors and use the hash values as IDs. However, hashing cannot retain semantic relationships between trained embeddings, such as distances learned in contrastive learning. Thus, we propose to use the cluster IDs to represent the embedding vectors instead. Specifically, we run a clustering algorithm[2] on all the embedding vectors, then we use the ID of the nearest cluster center as the ID feature for each embedding vector. There are some benefits of using cluster IDs: First, the clustering algorithm is an efficient approximation method that are scalable

to large data. Second, the cluster centers learned by the clustering algorithm have clear interpretations and retain the global and local distance structures. Third, since we are using Euclidean distance in the clustering algorithm, the learned cluster IDs can retain most of the distance information learned in the contrastive pre-training stage.

From Table. (1) we can conclude that the Cluster ID method consistently outperforms the Vector and SimScore method by a significant gap. The overall trend of SimScore and Cluster ID are similar, but evaluating with SimScore is much faster. Thus, practically we use the SimScore method to roughly compare different hyper-parameters, and the most promising hyper-parameters are further evaluated with the cluster ID method.

Why Cluster-ID is suitable for contrastive pre-training methods. The cosine similarity is related with the Euclidean distance as follows:

$$Sim(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}\,\mathbf{y}} = \frac{\mathbf{x}^2 + \mathbf{y}^2 - \mathbf{x} - \mathbf{y}^2}{2\mathbf{x}\,\mathbf{y}} \tag{2}$$

If we constrain the embedding vectors to be $\ell 2$ normalized, i.e., $\mathbf{x} = \mathbf{y} = 1$, which is the same as in contrastive learning. Then we have

$$Sim(\mathbf{x}, \mathbf{y}) = \frac{2 - \mathbf{x} - \mathbf{y}^2}{2} \tag{3}$$

Thus, maximizing the cosine similarity is equivalent to minimizing the Euclidean distance between $\ell 2$ normalized vectors. Since we are optimizing the cosine similarity of the embedding vectors, we are indeed optimizing their Euclidean distances. And such distance information is retained by the clustering algorithm using Euclidean distances. By adjusting the number of clusters, we can also change the information to be retained. To conclude, the Cluster-ID method aligns with both the pre-training and downstream stages, resulting in better performance.

2 Why not use random masked prediction and self-attention?

Random masked prediction is a popular pre-training style, which may also be applied to our framework. However, considering the data-generating process, random masking is indeed unnecessary and may cause some redundant computation and information leakage. Because of the time series nature of user click history, we will train on all the possible reconstruction targets in click history. That is, we will train with (B; A) at first, where (A) is the history and B is the target. After the user has clicked on item C, we will train on (C; B, A), and so on. Thus, every item in the click history will be treated as a target exactly once.

Suppose we adopted the random masking method, there is a chance that we train on (A; C, B), then we train on (C; A, B). Since the model has already

observed the co-occurrence of A, B, and C, the next (C; A, B) prediction will be much easier. The random masking method also violates the potential causation that observing A and B causes the user's interest in C.

Masked predictions typically use self-attention instead of cross attention with knowledge about the target as our proposed user intention reconstruction method. We also try with the self-attention method in the experiment section, which performs worse than our method. The reason is that learning to predict the next item with only a few click history is tough, but training to figure out how the next item can be reconstructed with the clicking history is much easier and can provide more meaningful signals during training.

3 Does projection head help?

Table 2 Adding projection to COURIER.

	AUC (women's clothing)	AUC	GAUC
w projection	0.29%	0.06%	-0.04%
COURIER	**0.46%**	**0.16%**	**0.19%**

Most of the contrastive pre-training methods are equipped with projection heads. A projection head refers to one or multiple layers (typically an MLP) that is appended after the last embedding layer. During training, the contrastive loss (e.g., InfoNCE) is calculated with the output of the projection head instead of using the embeddings directly. After training, the projection heads are dropped, and the embeddings are used in downstream tasks, e.g, classification. It is widely reported that training with projection heads can improve downstream performance[3–5]. However, the reason for the success of the projection head is still unclear. We experiment with projection heads as an extension of COURIER, and the results are in Table. (2). We find that projection heads have a negative impact on COURIER. There are two possible reasons: 1. In those contrastive methods, the downstream usage and the pre-training stage are inconsistent. In pre-training, the contrastive loss pushes embeddings and their augmented embeddings to be closer, which wipes away detailed information other than distance, while the distance itself cannot be used in classification. By adding projection heads, the projection head can learn the distance information without wiping away much detailed information. 2. In our COURIER method, we use the distance information learned in the pre-training stage directly by calculating the similarity score or cluster ID. Thus, the contrastive loss is consistent with the downstream usage. If we train with a projection head, the embeddings are not well suited for calculating the similarity scores since the embeddings are not trained to learn cosine similarity directly.

4 Implementation and Reproduction

Image Backbone. We adopt the swin-tiny implementation in Torchvision. The output layer of the backbone model is replaced with a randomly initialized linear layer, and we use weights pre-trained on the ImageNet dataset to initialize other layers. We apply gradient checkpoint[6] on all the attention layers in swin-tiny to reduce memory usage and enlarge batch size. We apply half-precision (float 16) computation[7] in all our models to accelerate computing and reduce memory. We train all the methods on a cluster with 48 Nvidia V100 GPUs (32GB), which enables single GPU batch size = 64, overall batch size = 64*48 = 3072. Note that each line of data contains 10 images, so the number of images within a batch is 30720 (some are padded zeros). The images are resized, cropped, and normalized before feeding to the backbone model. The input image size is 224.

We use the following hyperparameters for all the methods. Learning rate=1e-4, embedding size=256, weight decay=1e-6. We use the Adam optimizer. We have tuned these hyperparameters roughly but did not find significantly better choices on specific methods.

Hyperparameters of Courier: We use $\tau = 0.05$ as reported in the paper.

Implementation of CLIP: We tune the τ within [0.1, 0.2, 0.5], and find that $\tau = 0.1$ performs best, corresponding to the reported results. The batch size of CLIP is 32*48 since we have to reduce the batch size to load the language model. The effective batch size of CLIP is 32*48*5, since we only have titles of the PV items in our dataset. The text model is adapted from Chinese-BERT[8] and is initialized with the pre-trained weights provided by the authors.

Implementation of SimCLR: We tune the τ with in [0.1, 0.2, 0.5], and find that $\tau = 0.2$ performs best, corresponding to the reported results. The effective batch size of SimCLR is 64*48*10. We implemented the same augmentation strategy as suggested by the SimCLR paper[3].

Implementation of SimSiam: The effective batch size of SimSiam is 64*48*10. The augmentation method is the same as SimCLR.

References

[1] Zhang, Z., Sheng, X., Zhang, Y., Jiang, B., Han, S., Deng, H., Zheng, B.: Towards understanding the overfitting phenomenon of deep click-through rate models. In: CIKM, pp. 2671–2680 (2022)

[2] Yi, J., Zhang, L., Wang, J., Jin, R., Jain, A.K.: A single-pass algorithm for efficiently recovering sparse cluster centers of high-dimensional data. In: ICML. JMLR Workshop and Conference Proceedings, vol. 32, pp. 658–666 (2014)

[3] Chen, T., Kornblith, S., Norouzi, M., Hinton, G.E.: A simple framework for contrastive learning of visual representations. In: ICML. Proceedings

of Machine Learning Research, vol. 119, pp. 1597–1607 (2020)

[4] Chen, T., Kornblith, S., Swersky, K., Norouzi, M., Hinton, G.E.: Big self-supervised models are strong semi-supervised learners. In: NeurIPS (2020)

[5] Chen, X., He, K.: Exploring simple siamese representation learning. In: CVPR, pp. 15750–15758 (2021)

[6] Chen, T., Xu, B., Zhang, C., Guestrin, C.: Training deep nets with sublinear memory cost. CoRR **abs/1604.06174** (2016) https://arxiv.org/abs/1604.06174

[7] Micikevicius, P., Narang, S., Alben, J., Diamos, G.F., Elsen, E., García, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., Wu, H.: Mixed precision training. In: ICLR (2018)

[8] Sun, Z., Li, X., Sun, X., Meng, Y., Ao, X., He, Q., Wu, F., Li, J.: Chinesebert: Chinese pretraining enhanced by glyph and pinyin information. In: ACL/IJCNLP, pp. 2065–2075 (2021)