AdaEnsemble: Learning Adaptively Sparse Structured Ensemble Network for Click-Through Rate Prediction

Yachen Yan[1,*], Liubo Li[1]

[1]Credit Karma, 760 Market Street, San Francisco, California, USA, 94012

Abstract

Learning feature interactions is crucial to success for large-scale CTR prediction in recommender systems and Ads ranking. Researchers and practitioners extensively proposed various neural network architectures for searching and modeling feature interactions. However, we observe that different datasets favor different neural network architectures and feature interaction types, suggesting that different feature interaction learning methods may have their own unique advantages. Inspired by this observation, we propose AdaEnsemble: a Sparsely-Gated Mixture-of-Experts (SparseMoE) architecture that can leverage the strengths of heterogeneous feature interaction experts and adaptively learns the routing to a sparse combination of experts for each example, allowing us to build a dynamic hierarchy of the feature interactions of different types and orders. To further improve the prediction accuracy and inference efficiency, we incorporate the dynamic early exiting mechanism for feature interaction depth selection. The AdaEnsemble can adaptively choose the feature interaction depth and find the corresponding SparseMoE stacking layer to exit and compute prediction from. Therefore, our proposed architecture inherits the advantages of the exponential combinations of sparsely gated experts within SparseMoE layers and further dynamically selects the optimal feature interaction depth without executing deeper layers. We implement the proposed AdaEnsemble and evaluate its performance on real-world datasets. Extensive experiment results demonstrate the efficiency and effectiveness of AdaEnsemble over state-of-the-art models. We open-source the TensorFlow implementation of AdaEnsemble: https://github.com/yanyachen/AdaEnsemble.

Keywords

CTR prediction, Recommendation System, Feature Interaction, Mixture of Experts, Dynamic Inference, Early Exiting, AutoML, Deep Neural Network

1. Introduction

Click-through rate (CTR) prediction model [1] is an essential component for the large-scale search ranking, online advertising and recommendation system [2, 3, 4, 5].

Many deep learning-based models have been proposed for CTR prediction problems in the industry. They have become dominant in learning the useful feature interactions of the mixed-type input in an end-to-end fashion[5]. While every existing method focuses on automatically modeling different types of feature interactions, there have been very few attempts to model different types of interactions jointly and dynamically, such that one model architecture can be directly applied to different types of datasets. We believe that the ensemble of various interaction modules to generate heterogeneous feature interactions can complement the non-overlapping knowledge learned through each interaction learning approach.

With the aim of accomplishing the stated objective, we propose AdaEnsemble: a Sparsely-Gated Mixture-of-Experts (SparseMoE) hierarchical architecture to ensemble different interaction learning modules and dynamically select optimal feature interaction depth. AdaEnsemble encompasses SparseMoE layers and the Depth Selecting Controller. Within each SparseMoE layer of AdaEnsemble, there is a collection of interaction learning experts, and a trainable gating network determines a sparse combination of these experts to use for each example. Within the Depth Selecting Controller, a trainable gating network will choose the feature interaction depth for each example and recursively propagate feature interaction representations through SparseMoE layers to the corresponding depth for computing the prediction. Through these conditional computation mechanisms, we enlarged the model capacity exponentially maintaining computational efficiency.

The main contributions of this paper can be summarized as follows:

- We designed a novel model architecture called AdaEnsemble to ensemble various types of feature interaction learning modules by Sparsely-Gated Mixture-of-Experts (SparseMoE). Through utilizing MoE layers recursively with residual connections and normalization, AdaEnsemble can model different types of interactions jointly and dynamically.
- We designed an efficient and effective Depth Selecting Controller to adaptively choose the optimal feature

Woodstock'22: ACM SIGKDD Conference on Knowledge Discovery and Data Mining, AdKDD Workshop 2023, August 6–10, 2023, Long Beach, CA

*Corresponding author.

yachen.yan@creditkarma.com (Y. Yan);

liubo.li@creditkarma.com (L. Li)

0000-0002-1213-4343 (Y. Yan); 0009-0006-9933-2436 (L. Li)

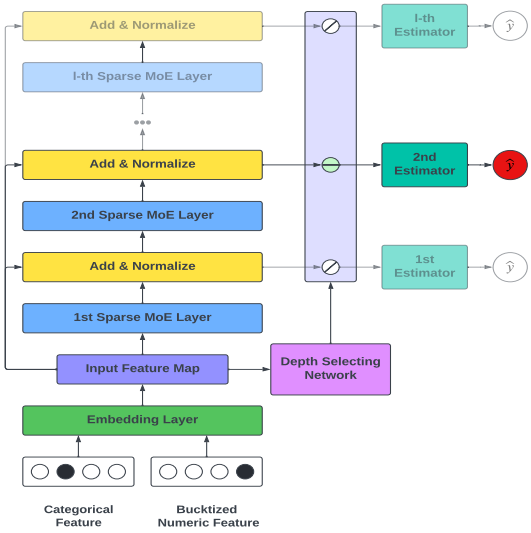

Figure 1: The Architecture of AdaEnsemble

In this example, the depth selecting network selects the 2nd layer to exit and compute the final prediction, therefore the deeper layers was not activated and plotted translucent in the figure.

interaction depth. Through utilizing this controller, AdaEnsemble can dynamically determine the layer for early exiting to improve prediction accuracy and inference efficiency.

- We applied a bi-level optimization algorithm for iteratively training the modeling network and gating network.

2. Proposed Model: AdaEnsemble

2.1. Feature Interaction Experts

We considered several types of feature interaction experts in our model: Dense Layer, Convolution Layer, Multi-Head Self-Attention Layer, Polynomial Interaction Layer, and Cross Layer. Essentially, any feature interaction learning layer can be included in our framework, and the residual connection and normalization will be applied to their ensembles. Now we introduce these feature interaction experts included in our framework. Note that our proposed framework is general and can use arbitrary feature interaction modules. The potential feature interaction experts can be used are not limited to the aforementioned.

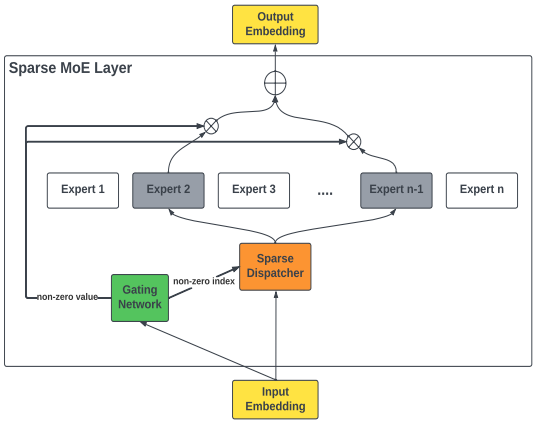

Figure 2: The architecture of Sparse Mixture-of-Experts Layer

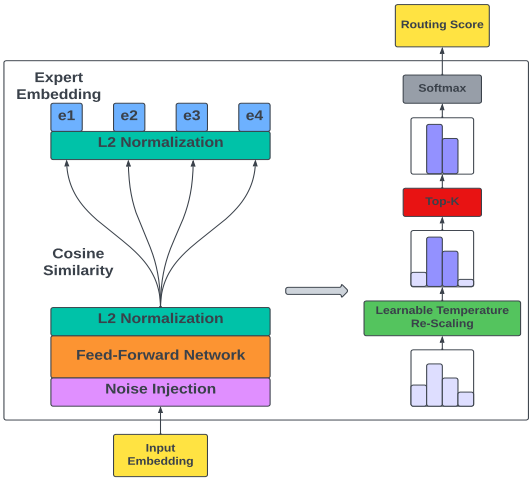

Figure 3: The Noisy Gating Network within Sparse Mixture-of-Experts Layer

2.2. Sparse Mixture-of-Experts Layer

The Sparse Mixture-of-Experts layer ensembles aforementioned heterogeneous feature interaction experts and consists of several other essential parts to make the overall model can be stably trained.

2.2.1. Noisy Gating Network

The gating network essentially computes the gating value for selecting experts for each input embedding and weighting the output embedding of each expert.

For the input embedding of gating network X_0, it firstly processed by the gating network: a two-layer feed-

forward network, i.e. a dimension reduction layer with reduction ratio r [6], a non-linear activation function and then a dense layer projecting to hidden state $h \in R^d$. Additionally, we applied multiplicative jitter noise for introducing exploration and promoting load balancing between different experts.

$$h = \text{FFN}(X_0 \circ \text{RandomUniform}(1.0 - \text{eps}, 1.0 + \text{eps})) \tag{1}$$

After projecting the input embedding to hidden state $h \in R^d$, we apply the L_2 normalization to both hidden state $h \in R^d$ and learnable expert embeddings $v_j \in R^d$, where j is the index of expert. Then, we compute the cosine similarity between the hidden state and expert embedding as the initial routing score. Here we encourage the uniformity of representations to avoid dominated experts issue.

$$s_j = \frac{h \cdot v_j}{\|h\|\|v_j\|} \tag{2}$$

Finally, we use a learnable temperature scalar τ to rescale the routing scores to the range $[-1, +1]$.

$$g_j = s_j / \tau \tag{3}$$

For the computed routing score g, we only keep the top k values and set the rest to $-\infty$, resulting in the corresponding softmax gating values equal 0. The j-th element of the output vector of the gating network is

$$ExpertG(X_0)_j = \frac{\exp\left(\text{TopK}(g,k)_j\right)}{\sum_{j=1}^{N} \exp\left(\text{TopK}(g,k)_j\right)}, \tag{4}$$

where

$$\text{TopK}(g,k)_j = \begin{cases} g_j & \text{if } g_j \text{ is in the top } k \text{ elements of } g \\ -\infty & \text{otherwise.} \end{cases} \tag{5}$$

These gating values will be used by the sparse dispatcher for routing examples to different experts. This is the essential step for achieving sparsity of our Sparse Mixture-of-Experts layer. Note that the $ExpertG(X_0)$ is differentiable regardless the value of k[7].

2.2.2. Annealing Top-K Gating

We also introduce annealing mechanism to the Top-K operation. We starts with k value equal to the number of experts, which means that we starts as a fully dense gate that routes examples to all experts. Then we gradually decrease the k and route examples to fewer experts, to adaptively make the structure sparser and continuously improving the computation efficiency.

By annealing of the k value, we start to train our architecture with a dense structure which allows us to thoroughly learn all experts and adjust the gating network in the correct direction at the beginning. Therefore, we can control the sparsity of our architecture while training to not only accelerate the convergence of the gating network but also benefit the experts' specialty for learning particular types of feature interactions.

2.2.3. Sparse Dispatcher

The sparse dispatcher [8, 7, 9] takes the examples gating values and experts as input. It firstly dispatches the examples to the experts corresponding to the non-zero gating values, and lets experts generate the output embeddings. The output y of the Sparse Mixture-of-Experts layer is the linearly weighted combination of expert output embeddings by the non-zero gating values.

$$y = \sum_{j \in \phi} ExpertG_j(X_0) E_j(X_0, X_l) \tag{6}$$

Where ϕ denotes the selected non-zero indices. We save computation based on the sparsity of $ExpertG(X_0)$. Wherever $ExpertG(X_0)_j = 0$, we don't pass the expert to the corresponding expert and do not need to compute expert output embedding $E_j(X_0, X_l)$.

2.2.4. Load Distribution Regularization

As stated in the previous research [8, 7, 9, 10], the gating network tends to select only a few experts if no regularization is applied, especially when certain experts are easier to train than other experts. This phenomenon is self-reinforcing, since the selected experts are trained more and will be selected more frequently by the gating network. Therefore, the load balancing loss is applied to enforce the uniform expert routing.

$$L_{\text{balance}} = \lambda \cdot N \cdot \sum_{j=1}^{N} f_j \cdot P_j \tag{7}$$

where B is the batch size, N is the number of experts, f_j is the fraction of examples dispatched to expert j, P_j is the average of the router probability allocated for expert j, and λ is the coefficient for the regularization term.

$$f_j = \frac{1}{B} \sum_{x \in \mathcal{B}} \mathbf{1}\{\text{argmax } p(x) = j\} \tag{8}$$

$$P_j = \frac{1}{B} \sum_{x \in \mathcal{B}} p_j(x) \tag{9}$$

While the default load balancing loss is applicable and effective when experts are of the same type, AdaEnsemble is using heterogeneous feature interaction experts, and the optimal load for each expert is not uniform. Therefore, we apply the below load distribution regularization

to encourage the expected load distribution of heterogeneous experts.

$$L_{\text{distribution}} = \lambda \cdot \sum_{j=1}^{N} \frac{f_j \cdot P_j}{w_j} \tag{10}$$

where hyper-parameter w_j is the expected load fraction of examples dispatched to expert j, and naturally $\sum_{j=1}^{N} w_j = 1$. In practice, the λ should be sufficiently large to prevent expert selection self-reinforcing phenomenon at the initial training stage while not overwhelming the primary LogLoss objective.

2.3. Depth Selecting Controller

2.3.1. Depth Selecting Network

The Depth Selecting Network is essentially the same configuration as the aforementioned Noisy Gating Network for SparseMoE layer. We denote it by $DepthG(X_0)$. The outputs of $DepthG(X_0)$ are $[g_1^{depth}, g_2^{depth}, \cdots, g_L^{depth}]$, indicating each example's optimal forward propagation depth. The l-th unit denotes the probability of selecting the l-th MoE layer to exit. The optimal depth is automatically selected as the one corresponding to the largest probability. In contrast to the expert selection, when choosing the optimal depth of each example for the dynamic inference, we only keep the top-1 depth index from the output units of the Depth Selecting Network. Note that we can also apply the load distribution regularization to encourage the examples' propagation depth distribution.

2.3.2. Dynamic Propagation Mechanism

With the depth gates $g_l^{depth} \in [0, 1]$ computed by Depth Selecting Network, we obtain the optimal depth for each example. If $g_l^{depth} = 0$, we recursively forward propagate examples through MoE layers and compute deeper representation until $g_l^{depth} = 1$ or reaching the final layer. If $g_l^{depth} = 1$, the forward propagation will be stopped and the corresponding l-th estimator will compute the prediction. To efficiently process a batch of examples with different optimal propagation depths, we utilize algorithm 1 for dynamic forward propagation.

2.4. Training

2.4.1. Training Objective

The loss function we use a linearly weighted combination of the Log Loss and the auxiliary load distribution regularization,

$$Loss = L_{\text{LogLoss}} + \lambda_1 L_{\text{distribution}}^{\text{expert}} + \lambda_2 L_{\text{distribution}}^{\text{depth}} \tag{11}$$

where λ_1 and λ_2 are the coefficients for weighting the load distribution regularization of experts and depth.

2.4.2. Bi-Level Optimization

The optimization task for training the AdaEnsemble is to jointly optimize the parameters W, which stands for the expert layers and estimator layers, and α, which represents the expert gating network and depth selecting network. Inspired by the DARTS [11], we apply bi-level optimization algorithm for training our model, where α is the upper-level parameters and W is the lower-level parameters. We apply algorithm 2 to optimize W and α alternatively and iteratively.

2.5. Discussion on AdaEnsemble

The combination of sparse experts routing within each SparseMoE layer and the early exiting by depth selecting controller brings two merits to the proposed model. On one hand, the stacked SparseMoE layers allow the proposed model to leverage the exponential combinations of sparsely gated experts, which brings in more predicting power. On the other hand, both the experts routing mechanism and the depth selecting mechanism enables the proposed model to learn the instance-ware expert combination and instance-ware model depth. These two conditional computation mechanisms improve the efficiency during model serving. In the next section, we will illustrate the effectiveness of the proposed model through some experimental studies.

3. Experiments

In this section, we focus on evaluating the effectiveness of our proposed models and seeking answers to the following research questions::

- **Q1**: How does our proposed AdaEnsemble perform compared to each baseline in the CTR prediction problem?
- **Q2**: How does the SparseMoE layer perform compared to DenseMoE, which utilizes all feature interaction experts? Does the cascade of SparseMoE layers effectively capture different types of feature interactions?
- **Q3**: How does the depth selecting controller perform compared to a full-depth network? Does the early exiting mechanism achieve both effectiveness and efficiency?

Algorithm 1 Dynamic Propagation

```
DepthGates ← DepthSelectingNetwork(X_0))
ŷ ← DynamicPropagation(X_0, DepthGates, depth=0)
return ŷ

function DynamicPropagation(Inputs, Gates, Depth)
    Outputs = MoE_Depth(Inputs)
    Depth += 1
    if Depth == Number of Layer then
        ŷ = Estimator_Depth(Outputs)
    else
        g = Gates[:, Depth]
        Outputs_keep, Outputs_exit = Dispatch(Outputs, g)
        Gates_keep, _ = Dispatch(Gates, g)

        ŷ_keep = DynamicPropagation(Outputs_keep, Gates_keep, Depth)
        ŷ_exit = Estimator_Depth(Outputs_exit)
        ŷ = Combine(ŷ_keep, ŷ_exit)
    end if
    return ŷ
end function
```

Algorithm 2 Bi-Level Optimization for AdaEnsemble

Input: training examples with labels, step size t
Output: well-learned parameters $\mathbf{W}^*$ and *

while not converged **do**
  Sample a mini-batch of validation data
  Updating by descending
  $\nabla \mathscr{L}_{val}(\mathbf{W} - \xi \nabla_{\mathbf{W}} \mathscr{L}_{train}(\mathbf{W},),)$
  ($\xi = 0$ for first-order approximation)
  for $i \leftarrow 1, t$ **do**
    Sample a mini-batch of training data
    Update $\mathbf{W}$ by descending $\nabla_{\mathbf{W}} \mathscr{L}_{train}(\mathbf{W},)$
  end for
end while

3.1. Experiment Setup

3.1.1. Datasets

We evaluate our proposed model on three public real-world datasets widely used for research.

1. Criteo.[1] Criteo dataset is from Kaggle competition in 2014. Criteo AI Lab officially released this dataset after, for academic use.

2. Avazu.[2] Avazu dataset is from Kaggle competition in 2015. Avazu provided 10 days of click-through data.

3. iPinYou.[3] iPinYou dataset is from iPinYou Global RTB(Real-Time Bidding) Bidding Algorithm Competition in 2013. We follow the data processing steps of [12].

[1]https://www.kaggle.com/c/criteo-display-ad-challenge
[2]https://www.kaggle.com/c/avazu-ctr-prediction
[3]http://contest.ipinyou.com/

Table 1
Performance Comparison of Different Algorithms on Criteo, Avazu and iPinYou Dataset.

	Criteo		Avazu		iPinYou	
Model	AUC	LogLoss	AUC	LogLoss	AUC	LogLoss
LR	0.7924	0.4577	0.7533	0.3952	0.7692	0.005605
FM	0.8030	0.4487	0.7652	0.3889	0.7737	0.005576
DNN	0.8051	0.4461	0.7627	0.3895	0.7732	0.005749
Wide&Deep	0.8062	0.4451	0.7637	0.3889	0.7763	0.005589
DeepFM	0.8069	0.4445	0.7665	0.3879	0.7749	0.005609
DeepCrossing	0.8068	0.4456	0.7628	0.3891	0.7706	0.005657
DCN	0.8056	0.4457	0.7661	0.3880	0.7758	0.005682
PNN	0.8083	0.4433	0.7663	0.3882	0.7783	0.005584
xDeepFM	0.8077	0.4439	0.7668	0.3878	0.7772	0.005664
AutoInt	0.8053	0.4462	0.7650	0.3883	0.7732	0.005758
FiBiNET	0.8082	0.4439	0.7652	0.3886	0.7756	0.005679
xDeepInt	0.8111	0.4408	0.7672	0.3876	0.7790	0.005567
DCN V2	0.8086	0.4433	0.7662	0.3882	0.7765	0.005593
AdaEnsemble	**0.8132**	**0.4394**	**0.7687**	**0.3865**	**0.7807**	**0.005550**

3.1.2. Competing Models

We compare AdaEnsemble with following models: LR (Logistic Regression) [13, 2], FM (Factorization Machine) [14], DNN (Multilayer Perceptron), Wide & Deep [4], DeepCrossing [15], DCN (Deep & Cross Network) [16], PNN (with both inner product layer and outer product layer) [17, 18], DeepFM [19], xDeepFM [20], AutoInt [21], FiBiNET [22], xDeepInt[23] and DCN V2 [24]. Some of the models are state-of-the-art models for CTR prediction problem and are widely used in the industry.

3.2. Model Performance Comparison (Q1)

The overall performance of different model architectures is listed in Table 1. We have the following observations in terms of model effectiveness:

- Models with more than two feature interaction mod-

ules generally perform better than models with only a single feature interaction module, indicating the importance of jointly learned feature interaction representation.

- The optimal feature interaction depth varies by feature interaction module type and when combined with different module types, indicating the necessity for dynamically combining different feature interactions on different interaction depths.
- AdaEnsemble achieves the best prediction performance among all models. Our model's superior performance could be attributed to the fact that AdaEnsemble jointly model various types of feature interactions by adaptively selecting the feature interaction experts combination and determining the optimal feature interaction depth by the controller.

3.3. Feature Interaction Expert Selection Analysis (Q2)

We compare the model performance and FLOPs between the DenseMoE and SparseMoE layers in AdaEnsemble architecture. We also include the performance of different multi-layer single expert models and their ensemble. All the performance of above methods are listed in Table 2. We also draw the alluvial diagram Figure 4 to illustrate the dependency of each SparseMoE layer's expert selection. The color of the flow is clustered by the frequency of the expert combination. Based on the above observations, we developed following understandings:

- Utilizing different feature interaction experts result in better performance than single expert models in general. SparseMoE layer achieves a better tradeoff between accuracy and computation efficiency.
- Only utilizing one expert per SparseMoE layer generally hurts the model performance as the model cannot ensemble different types of feature interactions.
- When utilizing more than one expert per SparseMoE layer, even though only a subset of feature interaction experts are selected, SparseMoE can still effectively capture the most significant feature interactions of different depths and maintain similar performance as the DenseMoE layer and superior performance to ensemble, while including more experts can also result in more computational cost.
- Figure 4 shows that the SparseMoE layers dynamically utilize a different combination of experts across different layers to capture the complex feature interactions effectively. That also explains why fusing different feature interactions is crucial for prediction accuracy.

Table 2
Performance Comparison of SparseMoE and DenseMoE on Criteo Dataset.

	AUC	LogLoss	FLOPs
SparseMoE(k=1)	0.8096	0.4423	2.26M
SparseMoE(k=2)	0.8121	0.4400	4.14M
SparseMoE(k=3)	0.8132	0.4394	6.02M
SparseMoE(k=4)	0.8133	0.4393	7.09M
DenseMoE	0.8133	0.4392	9.78M
Ensemble	0.8120	0.4401	12.15M
Dense Expert Only	0.8050	0.4463	3.71M
Cross Expert Only	0.8086	0.4433	3.36M
Polynomial Expert Only	0.8111	0.4408	3.32M
CNN Expert Only	0.8022	0.4501	1.11M
MHSA Expert Only	0.8051	0.4465	2.17M

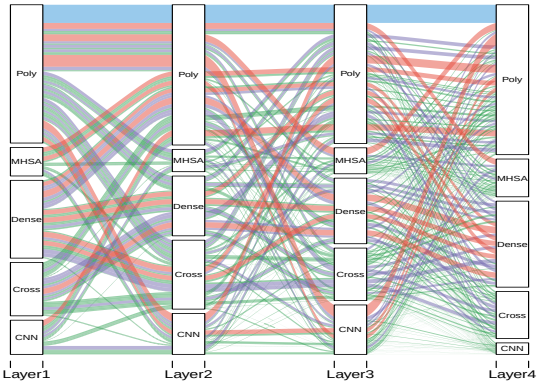

Figure 4: The Alluvial diagram for illustrating the dependency of each SparseMoE layer's expert selection

Each vertical axis represents a SparseMoE layer and the proportion of an expert being used. The horizontal flows indicate the dependency and relation of different SparseMoE layer's expert selection. The proportion of the expert combination was represented by the width of the flows and further clustered to different colors.

3.4. Depth Selection Analysis (Q3)

We compare the model performance between the AdaEnsemble with and without depth selecting controller to investigate whether the model achieves the harmony between prediction accuracy and inference efficiency with respect to depth selection. The performance of the different types of MoE layers and ensemble result is listed in Table 3.

With the incorporation of the depth selecting controller, we can observe that our model can significantly improve training complexity and inference efficiency (measured in FLOPs) while achieving slightly better performance than the full-depth model. We think the full-depth model is easier to overfit compared to AdaEnsemble, thus resulting in slightly worse accuracy perfor-

Table 3
Performance Comparison of AdaEnsemble with and without controller on Criteo Dataset.

	AUC	LogLoss	FLOPs
w/ controller	0.8132	0.4394	6.02M
w/o controller	0.8128	0.4396	8.58M

Table 4
AdaEnsemble Propagation Depth on Criteo Dataset.

	Layer 1	Layer 2	Layer 3	Layer 4
Fraction	6.53%	19.36%	66.43%	7.68%

mance. The AdaEnsemble with depth selecting controller adaptively selects feature interaction depth per example basis, thus achieving better trade-offs between prediction accuracy and inference efficiency. The distribution of per example forward propagation depth is listed in Table 4.

4. Conclusion

In this paper, we present a novel model architecture to click-through rate (CTR) modeling by introducing the Sparse-Gated Mixture-of-Experts (SparseMoE) hierarchical architecture for ensemble learning of heterogeneous feature interactions experts. A Depth Selecting Controller component was integrated into the model to dynamically select the optimal feature interaction depth for each instance. The utilization of these two conditional computation mechanisms results in a model architecture that can select a subset of feature interactions experts and the optimal interaction depth for each instance simultaneously, leading to an exponential increase in model capacity without incurring a corresponding increase in inference cost. Our extensive experiment demonstrate the superiority of our approach in terms of effectiveness and efficiency.

Future work will be dedicated to exploring the potential for extending our method to the modeling of user behavior sequences. By learning a sparse ensemble of models, we anticipate that our approach can dynamically select the optimal expert for different behaviors in the context of user behavior sequence data.

References

[1] M. Richardson, E. Dominowska, R. Ragno, Predicting clicks: estimating the click-through rate for new ads, in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 521–530.

[2] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al., Ad click prediction: a view from the trenches, in: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2013, pp. 1222–1230.

[3] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, et al., Practical lessons from predicting clicks on ads at facebook, in: Proceedings of the Eighth International Workshop on Data Mining for Online Advertising, ACM, 2014, pp. 1–9.

[4] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al., Wide & deep learning for recommender systems, in: Proceedings of the 1st workshop on deep learning for recommender systems, ACM, 2016, pp. 7–10.

[5] S. Zhang, L. Yao, A. Sun, Y. Tay, Deep learning based recommender system: A survey and new perspectives, ACM Computing Surveys (CSUR) 52 (2019) 5.

[6] J. Hu, L. Shen, G. Sun, Squeeze-and-excitation networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 7132–7141.

[7] W. Fedus, B. Zoph, N. Shazeer, Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

[8] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, J. Dean, Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, arXiv preprint arXiv:1701.06538 (2017).

[9] B. Zoph, I. Bello, S. Kumar, N. Du, Y. Huang, J. Dean, N. Shazeer, W. Fedus, Designing effective sparse expert models, arXiv preprint arXiv:2202.08906 (2022).

[10] Z. Chi, L. Dong, S. Huang, D. Dai, S. Ma, B. Patra, S. Singhal, P. Bajaj, X. Song, F. Wei, On the representation collapse of sparse mixture of experts, arXiv preprint arXiv:2204.09179 (2022).

[11] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, arXiv preprint arXiv:1806.09055 (2018).

[12] W. Zhang, S. Yuan, J. Wang, X. Shen, Real-time bidding benchmarking with ipinyou dataset, arXiv preprint arXiv:1407.7073 (2014).

[13] H. B. McMahan, Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization (2011).

[14] S. Rendle, Factorization machines, in: 2010 IEEE International Conference on Data Mining, IEEE, 2010, pp. 995–1000.

[15] Y. Shan, T. R. Hoens, J. Jiao, H. Wang, D. Yu, J. Mao, Deep crossing: Web-scale modeling without manually crafted combinatorial features, in: Proceedings

of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 255–262.

[16] R. Wang, B. Fu, G. Fu, M. Wang, Deep & cross network for ad click predictions, in: Proceedings of the ADKDD'17, ACM, 2017, p. 12.

[17] Y. Qu, H. Cai, K. Ren, W. Zhang, Y. Yu, Y. Wen, J. Wang, Product-based neural networks for user response prediction, in: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, 2016, pp. 1149–1154.

[18] Y. Qu, B. Fang, W. Zhang, R. Tang, M. Niu, H. Guo, Y. Yu, X. He, Product-based neural networks for user response prediction over multi-field categorical data, ACM Transactions on Information Systems (TOIS) 37 (2018) 5.

[19] H. Guo, R. Tang, Y. Ye, Z. Li, X. He, Deepfm: a factorization-machine based neural network for ctr prediction, arXiv preprint arXiv:1703.04247 (2017).

[20] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, G. Sun, xdeepfm: Combining explicit and implicit feature interactions for recommender systems, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 1754–1763.

[21] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, J. Tang, Autoint: Automatic feature interaction learning via self-attentive neural networks, arXiv preprint arXiv:1810.11921 (2018).

[22] T. Huang, Z. Zhang, J. Zhang, Fibinet: Combining feature importance and bilinear feature interaction for click-through rate prediction, arXiv preprint arXiv:1905.09433 (2019).

[23] Y. Yan, L. Li, xdeepint: a hybrid architecture for modeling the vector-wise and bit-wise feature interactions (2020).

[24] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, E. Chi, Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems, in: Proceedings of the Web Conference 2021, 2021, pp. 1785–1797.