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

YaChen Yan, Liubo Li

August 2, 2023
Summary

1. Introduction

2. Model Architecture
   - Overview
   - Sparse Mixture-of-Experts Layer
   - Depth Selecting Controller
   - Training
   - Recap

3. Experiment

4. Conclusions
Introduction
Learning feature interactions are important to the model performance of online ads ranking system.

Extensive efforts have been introduced to learn different types of feature interactions: DeepCrossing, DeepFM, PNN, xDeepFM, AutoInt, FiBiNET, xDeepInt, DCN V2.
The practical performance of those designs can vary by dataset.

Different feature interaction learning methods may have different advantages and the interactions captured by them have non-overlapping information.
Motivation

- **Ensemble**
  - Ensemble of various interaction modules to generate heterogeneous feature interactions.
  - Each feature interaction learning approach can complement the non-overlapping knowledge.

- **Conditional Computation (instance-aware)**
  - Dynamically select feature interaction types.
  - Dynamically select optimal feature interaction depth.
Model Architecture
The Architecture of AdaEnsemble

Input Feature Map → Embedding Layer

- Categorical Feature
- Bucktized Numeric Feature

Embedding Layer → Add & Normalize

Add & Normalize → 1st Sparse MoE Layer

1st Sparse MoE Layer → Add & Normalize

Add & Normalize → 2nd Sparse MoE Layer

2nd Sparse MoE Layer → Add & Normalize

Add & Normalize → I-th Sparse MoE Layer

I-th Sparse MoE Layer → Add & Normalize

Add & Normalize → Depth Selecting Network

Depth Selecting Network → I-th Estimator

I-th Estimator → \( \hat{y} \)

Depth Selecting Network → 2nd Estimator

2nd Estimator → \( \hat{y} \)

Depth Selecting Network → 1st Estimator

1st Estimator → \( \hat{y} \)
AdaEnsemble Components

- **Sparse Mixture-of-Experts Layer:**
  - Feature Interaction Experts
  - Selecting the feature interaction experts
  - Dynamically (instance-aware) activate and ensemble them

- **Depth Selecting Controller**
  - Selecting the feature interaction depth
  - Recursively forward-propagate and compute deeper feature interactions
  - Compute final prediction when reaching selected depth (instance-aware)
The Architecture of Sparse Mixture-of-Experts Layer

Sparse MoE Layer

Output Embedding

Gating Network

Sparse Dispatcher

Input Embedding

Expert 1

Expert 2

Expert 3

... Expert n-1

Expert n

non-zero index

non-zero value
SparseMoE Components

- Feature Interaction Experts
- Noisy Gating Network:
  - A neural network selecting the Top-K experts per instance.
  - Annealing Top-K Gating.
  - Load Distribution Regularization.
- Sparse Dispatcher
  - Dispatch input and sparsely activate corresponding experts.
  - Combine each expert’s output.
Feature Interaction Experts

- **Dense Layer**
  \[ X_l = \sigma(W_l \cdot X_{l-1}) \]

- **Convolution Layer**
  \[ X_l = \text{Dense}(\text{Pooling}(\text{Conv1D}(\text{Reshape}(X_{l-1})))) \]

- **Multi-Head Self-Attention Layer**
  \[ X_l = \text{Dense}(\text{MultiHeadSelfAttention}(\text{Reshape}(X_{l-1}))) \]

- **Polynomial Interaction Layer**
  \[ X_l = X_{l-1} \circ (W_l \cdot X_0) \]

- **Cross Layer**
  \[ X_l = X_0 \circ (W_l \cdot X_{l-1}) + b_l \]
Noisy Gating Network

Figure: The Noisy Gating Network within Sparse Mixture-of-Experts Layer
Annealing Top-K Gating:
- Dense $\rightarrow$ Sparse.
- All experts $\rightarrow$ Fewer experts.
- Fully trained experts $\rightarrow$ Expert Routing.

Load Distribution Regularization
- Homogeneous Experts: $L_{\text{balance}} = \lambda \cdot N \cdot \sum_{j=1}^{N} f_j \cdot P_j$.
  - $f_j$ is the fraction of examples dispatched to expert $j$
  - $P_j$ is the average of the router probability allocated for expert $j$
- Heterogeneous Experts: $L_{\text{distribution}} = \lambda \cdot \sum_{j=1}^{N} \frac{f_j \cdot P_j}{w_j}$.
  - $\sum_{j=1}^{N} w_j = 1$
Depth Selecting Controller

- **Estimator Layer:**
  - Different interaction depth has a corresponding estimator layer.
  - A dense layer computes the final prediction.

- **Depth Selecting Network:**
  - A neural network selects the feature interaction depth per instance.
  - **Recursive Propagation**
    - Compute deeper feature interactions (Enter).
    - Compute final predictions (Exit)
Figure: Visualization of Recursive Propagation
Loss Function

\[
Loss = L_{\text{LogLoss}} + \lambda_1 L_{\text{distribution}}^{\text{expert}} + \lambda_2 L_{\text{distribution}}^{\text{depth}}
\] (1)

where \( \lambda_1 \) and \( \lambda_2 \) are the coefficients for weighting the load distribution regularization of experts and depth.
Bi-Level Optimization

- Bi-Level Optimization Algorithm:
  - Iteratively optimize the parameters $W$ and $\alpha$.
  - $W$: the expert layers and estimator layers.
  - $\alpha$: the expert gating network and depth selecting network.

Algorithm 2: Bi-Level Optimization for AdaEnsemble

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

1: while not converged do
2:   Sample a mini-batch of validation data
3:   Updating $\alpha$ by descending $\nabla_\alpha L_{val}(W - \xi \nabla_w L_{train}(W, \alpha), \alpha)$
4:   ($\xi = 0$ for first-order approximation)
5:   for $i \leftarrow 1, t$ do
6:     Sample a mini-batch of training data
7:     Update $W$ by descending $\nabla_w L_{train}(W, \alpha)$
8:   end for
9: end while
Recap
Experiment
## Model Performance Comparison

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

*YaChen Yan, Liubo Li, August 2, 2023, 21/26*
Table: Performance Comparison of SparseMoE and DenseMoE on Criteo Dataset.

<table>
<thead>
<tr>
<th>Expert Configuration</th>
<th>AUC</th>
<th>LogLoss</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>SparseMoE(k=1)</td>
<td>0.8096</td>
<td>0.4423</td>
<td>2.26M</td>
</tr>
<tr>
<td>SparseMoE(k=2)</td>
<td>0.8121</td>
<td>0.4400</td>
<td>4.14M</td>
</tr>
<tr>
<td>SparseMoE(k=3)</td>
<td>0.8132</td>
<td>0.4394</td>
<td>6.02M</td>
</tr>
<tr>
<td>SparseMoE(k=4)</td>
<td>0.8133</td>
<td>0.4393</td>
<td>7.09M</td>
</tr>
<tr>
<td>DenseMoE</td>
<td>0.8133</td>
<td>0.4392</td>
<td>9.78M</td>
</tr>
<tr>
<td>Ensemble</td>
<td>0.8120</td>
<td>0.4401</td>
<td>12.15M</td>
</tr>
<tr>
<td>Dense Expert Only</td>
<td>0.8050</td>
<td>0.4463</td>
<td>3.71M</td>
</tr>
<tr>
<td>Cross Expert Only</td>
<td>0.8086</td>
<td>0.4433</td>
<td>3.36M</td>
</tr>
<tr>
<td>Polynomial Expert Only</td>
<td>0.8111</td>
<td>0.4408</td>
<td>3.32M</td>
</tr>
<tr>
<td>CNN Expert Only</td>
<td>0.8022</td>
<td>0.4501</td>
<td>1.11M</td>
</tr>
<tr>
<td>MHSA Expert Only</td>
<td>0.8051</td>
<td>0.4465</td>
<td>2.17M</td>
</tr>
</tbody>
</table>
Figure: The Alluvial diagram for illustrating the dependency of each SparseMoE layer's expert selection.
Conclusions
Conclusions

- We propose a AdaEnsemble leveraging a Sparse-Gated Mixture-of-Experts (SparseMoE) layer and a Depth Selecting Controller, which increased model capacity without raising online inference cost.
- With the conditional computation mechanism applied, the model selects feature interaction experts and optimal depth for each example simultaneously.
Thank You!