Towards Open-World Feature Extrapolation: An Inductive Graph Learning Approach

Qitian Wu, Chenxiao Yang, Junchi Yan
Department of Computer Science and Engineering
Shanghai Jiao Tong University
General problem: learn a mapping from input features to labels

- Input data $x = [x_1, x_2, \cdots, x_d]$ where $x_i$ denotes the i-th input feature
- Assume a prediction model $f : x \rightarrow y$ and objective

$$f^* = \arg \min_f \mathbb{E}_{(x,y) \sim D}[l(f(x), y)]$$

Applications

- Tabular data: weather/income/usage prediction, disease diagnosis…
- Real systems: recommendation, advertisement, question answering…

**Scenario 1:**
Predict a person’s income with age/occ/edu

<table>
<thead>
<tr>
<th>age</th>
<th>occ</th>
<th>edu</th>
<th>income</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_1$</td>
<td>$x_{11}$</td>
<td>$x_{12}$</td>
<td>$x_{13}$</td>
</tr>
<tr>
<td>$o_2$</td>
<td>$x_{21}$</td>
<td>$x_{22}$</td>
<td>$x_{23}$</td>
</tr>
<tr>
<td>$o_3$</td>
<td>$x_{31}$</td>
<td>$x_{32}$</td>
<td>$x_{33}$</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

**Scenario 2:**
Predict whether a user would click an item with attributes

Amazon.com

- user features: age/gender...
- item features: category/price…

Recommended for You

- [The Little Big Things: 162 Ways to Pursue BIGDREAMS](https://example.com)
- [Harlan Ellison's Perception and Consciousness](https://example.com)
- [Sherlock Holmes: [Holmes]](https://example.com)
- [Alas, Poor Yorick](https://example.com)
Challenges and Limitations of Neural Networks

- **Challenges** for attribute feature learning
  - New features *dynamically* appear (unseen features in test set)
  - Scenarios: heterogeneous data sources, multi-modal data

- **How can neural networks deal with new features**
  - Retraining from scratch
    - Issue: time-consuming
  - Incremental learning on new features
    - Issue: over-fitting & catastrophic forgetting

- **Inductive reasoning ability**
  - Humans possess inherent ability for understanding new information
Preprocessing: convert raw inputs to multi-hot vectors

- Raw input $r_i = [r_{i1}, r_{i2}, \cdots, r_{id}]$ where $r_{im}$ denotes the $m$-th raw feature
- For categorical feature: one-hot encoding representation
- For continuous feature: first discretization then one-hot encoding

$$x_i = [x_i^1, x_i^2, \cdots, x_i^d]$$ where $x_i^m$ is a one-hot vector

Open-world feature extrapolation:

- Training Set
  - Data $\{(x_i, y_i)\}_{i \in I_{tr}}$
  - Feature space $x_i \in \mathcal{X}_{tr} = \{0, 1\}^D$
  - Label space $y_i \in \mathcal{Y}$

- Test Set
  - Data $\{(x_i', y_i')\}_{i' \in I_{te}}$
  - Feature space $x_i' \in \mathcal{X}_{te} = \{0, 1\}^{D'}$
  - Label space $y_i' \in \mathcal{Y}$

Two cases causing feature space expansion:

1) new raw features come, 2) unseen feature values out of known range
Key Observation 1: Permutation-Invariance

- Neural networks can be decomposed into two parts

\[ \hat{y}_i = h(x_i; \phi, W) \]
\[ z_i = Wx_i \]
\[ \hat{y}_i = \text{FFN}(z_i; \phi) \]

- Equivalent view: feature embedding look-up + embedding aggregation

Key insight: The permutation-invariance property enables variable-length input features
Key Observation 2: Feature-Data Graph

- The input feature-data matrix can be treated as a **bipartite graph**

\[
X_{tr} = [x_i]_{i \in I_{tr}} \in \{0, 1\}^{N \times D}
\]

- **Feature nodes** \( F_{tr} = \{f_j\}_{j=1}^{D} \)
- **Instance nodes** \( I_{tr} = \{o_i\}_{i=1}^{N} \)
- **Adjacency matrix** \( X_{tr} \)

Advantage of graph representation:
1) **Variable-size** for features/instances
2) **Missing** values are allowed

**Key insight:** Convert inferring embeddings for new features to inductive representation on graphs

---

**Observed Data Matrix**

<table>
<thead>
<tr>
<th></th>
<th>( f_1 )</th>
<th>( f_2 )</th>
<th>( f_3 )</th>
<th>( f_4 )</th>
<th>( f_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( o_1 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( o_2 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( o_3 )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( o_4 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Feature-Data Graph**
Proposed Model Framework: FATE

- **High-level GNN**: take feature-data matrix as input and update feat. embeddings
- **Low-level backbone**: take each instance as input and output prediction
Details for Proposed Model

- **GNN model feedforward**
  - Feature nodes \( \{w_j\}_{j=1}^D \) (initial embeddings as \( w_j^{(0)} \))
  - Instance nodes \( \{s_i\}_{i=1}^N \) (initial states \( s_i^{(0)} = 0 \))
  - Message passing rule:
    \[
    a_i^{(l)} = \text{AGG}\left(\{w_k^{(l-1)} | \forall k, x_{ik} = 1\}\right) \\
    s_i^{(l)} = P^{(l)} \text{COMB}\left(s_i^{(l-1)}, a_i^{(l-1)}\right) \\
    b_j^{(l)} = \text{AGG}\left(\{s_k^{(l-1)} | \forall k, x_{jk} = 1\}\right) \\
    w_j^{(l)} = P^{(l)} \text{COMB}\left(w_j^{(l-1)}, b_j^{(l-1)}\right)
    \]
Entire feedforward compute

- Query feature embeddings
  - For old features: $W$
  - For new features: set as zero
- Update feature embeddings
  \[ \hat{W} = [w_j^{(L)}]_{j=1}^{D} = g(W, X; \omega) \]
- Assign to backbone and output predicted results
  \[ \hat{y}_i = h(x_i; \phi, \hat{W}) \]

Note: 1) $X$ can be either training or test data; 2) the permutation-invariance and graph representation enables arbitrarily sized $X$
Proposed Training Approach

- Two useful techniques for learning to extrapolate
  - Proxy training data
    - Self-supervised learning: n-fold splitting input features
    - Inductive learning: k-shot sampling input features
  - Asynchronous Updates
    - Fast/slow for backbone/GNN

- DropEdge regularization
- Scaling to large systems
  - Mini-batches along the instance dimension (complexity $O(Bd)$)
Generalization Error Analysis

- **Key aspect:** we treat input data matrix as a whole and the proposed proxy data-based training approach samples data point from
  \[ S = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_M, Y_M)\} \quad \text{where} \quad M \sim \mathcal{O}\left(\frac{d!}{(d-k)!k!}\right) \]

- **The empirical risk** over training data
  \[ R_{emb}(h_S) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}(Y_m, h(X_m; \psi_S)) \]

- **The generalization error** can be defined as
  \[ R(h_S) = \mathbb{E}_{(X,Y)}[\mathcal{L}(Y, h(X; \psi_S))] \]

- **We care about expected generalization gap** over random sampling
  \[ \mathbb{E}_A[R(h_S) - R_{emp}(h_S)] \]
Theorem. Assume the loss function is bounded by $l(y_i, \hat{y}_i) \leq \lambda$. For a learning algorithm trained on data $\{X_{tr}, Y_{tr}\}$ with $T$ iterations of SGD updates, with probability at least $1 - \delta$, we have

$$\mathbb{E}_A[R(h_S) - R_{emp}(h_S)] \leq \mathcal{O}\left(\frac{d^T}{M}\right) + \left(\mathcal{O}\left(\frac{d^T}{M^2} + \lambda\right)\sqrt{\frac{\log(1/\delta)}{2M}}\right)$$

where $M \propto \mathcal{O}\left(\frac{d!}{(d - k)!k!}\right)$ and $k$ denotes the size of sampled features.

Note: 1) The generalization gap depends on the number of raw features, i.e. $d$

2) The size $M$ is determined by the configuration of proxy training data. (If there is more randomness, then $M$ would be larger)

Is larger $M$ always better? No! larger variance and larger optimization error
Experiments on UCI Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Domain</th>
<th>#Instances</th>
<th>#Raw Feat.</th>
<th>Cardinality</th>
<th>#0-1 Feat.</th>
<th>#Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gene</td>
<td>Life</td>
<td>3190</td>
<td>60</td>
<td>4~6</td>
<td>287</td>
<td>3</td>
</tr>
<tr>
<td>Protein</td>
<td>Life</td>
<td>1080</td>
<td>80</td>
<td>2~8</td>
<td>743</td>
<td>8</td>
</tr>
<tr>
<td>Robot</td>
<td>Computer</td>
<td>5456</td>
<td>24</td>
<td>9</td>
<td>237</td>
<td>4</td>
</tr>
<tr>
<td>Drive</td>
<td>Computer</td>
<td>58509</td>
<td>49</td>
<td>9</td>
<td>378</td>
<td>11</td>
</tr>
<tr>
<td>Calls</td>
<td></td>
<td>7195</td>
<td>10</td>
<td>4~10</td>
<td>219</td>
<td>10</td>
</tr>
<tr>
<td>Github</td>
<td>Social</td>
<td>37700</td>
<td>-</td>
<td>~</td>
<td>4006</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Evaluation**: training on observed features and testing on all features
  - **Instance-level**: random split all the instances into training/validation/test data
  - **Feature-level**: random split all the features into observed/unobserved features

- **Baselines/Competitors**:
  - **Base**: (use observed features for tr/te), **Oracle**: (use all features for tr)
  - **Simple extrapolation approaches**: Avg, KNN, Mean pooling
  - **Incremental learning**: (first train on observed feat, then retrain on unobserved)

- **Implementation**: 3-layer NN as backbone, 4-layer GNN
Experiments on UCI Datasets

Results:
1) FATE (ours) yields 7.3% higher acc. and 1.3% higher AUC than Base
2) FATE achieves very close performance to Oracle (using all features)
3) FATE produces 29.8% higher acc. than baselines Avg, KNN, Pooling
3) FATE even outperforms INL in most cases with averagely 4.1% impv.
Experiments on UCI Datasets

Key insights:
1) FATE’s produced embeddings for observed/unobserved features have dissimilar distributions compared to Oracle.
   - **FATE manages to extract some informative knowledge from new features**
2) The embeddings of FATE form some particular structures.
   - **FATE could further capture feature-level relations through GNN interaction**

Figure. T-SNE visualization of feature embeddings produced by FATE (ours) and Oracle. Red for observed features and yellow for unobserved ones.
Experiments on Advertisement Click Prediction

- **Scenario**: click-through rate (CTR) prediction for online advertisement
  - **Goal**: predict whether a user would click on a displayed ad. (binary classification)
  - **Input**: attribute features for users/ads
    - Typical features: device id, site id, app id, ad category, app category, etc.
    - The **ID features** have massive values which induces large feature dimensions

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Domain</th>
<th>#Instances</th>
<th>#Raw Feat.</th>
<th>Cardinality</th>
<th>#0-1 Feat.</th>
<th>#Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avazu</td>
<td>Ad.</td>
<td>40,428,967</td>
<td>22</td>
<td>5~1611749</td>
<td>2,018,025</td>
<td>2</td>
</tr>
<tr>
<td>Criteo</td>
<td>Ad.</td>
<td>45,840,617</td>
<td>39</td>
<td>5~541311</td>
<td>2,647,481</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Evaluation**: chronologically split all the instances into 10-fold
  - Use first subset for training, second for validation and the remaining for test
  - ~1.3M/~0.4M/~0.8M exclusive features in training/validation/test data in Criteo

- **Implementation**: 3-layer NN/DeepFM as backbones
Experiments on Advertisement Click Prediction

Table. ROC-AUC results for eight test sets (T1 - T8) on Avazu and Criteo

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Backbone</th>
<th>Model</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avazu</td>
<td>NN</td>
<td>Base</td>
<td>0.666</td>
<td>0.680</td>
<td>0.691</td>
<td>0.694</td>
<td>0.699</td>
<td>0.703</td>
<td>0.705</td>
<td>0.705</td>
<td>0.693 ± 0.012</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pooling</td>
<td>0.655</td>
<td>0.671</td>
<td>0.683</td>
<td>0.683</td>
<td>0.689</td>
<td>0.694</td>
<td>0.697</td>
<td>0.697</td>
<td>0.684 ± 0.011</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FATE</td>
<td><strong>0.689</strong></td>
<td><strong>0.699</strong></td>
<td><strong>0.708</strong></td>
<td><strong>0.710</strong></td>
<td><strong>0.715</strong></td>
<td><strong>0.720</strong></td>
<td><strong>0.721</strong></td>
<td><strong>0.721</strong></td>
<td><strong>0.710 ± 0.010</strong></td>
</tr>
<tr>
<td></td>
<td>DeepFM</td>
<td>Base</td>
<td>0.675</td>
<td>0.684</td>
<td>0.694</td>
<td>0.697</td>
<td>0.699</td>
<td>0.706</td>
<td>0.708</td>
<td>0.706</td>
<td>0.697 ± 0.009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pooling</td>
<td>0.666</td>
<td>0.676</td>
<td>0.685</td>
<td>0.685</td>
<td>0.688</td>
<td>0.693</td>
<td>0.694</td>
<td>0.694</td>
<td>0.685 ± 0.009</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FATE</td>
<td><strong>0.692</strong></td>
<td><strong>0.702</strong></td>
<td><strong>0.711</strong></td>
<td><strong>0.714</strong></td>
<td><strong>0.718</strong></td>
<td><strong>0.722</strong></td>
<td><strong>0.724</strong></td>
<td><strong>0.724</strong></td>
<td><strong>0.713 ± 0.010</strong></td>
</tr>
<tr>
<td>Criteo</td>
<td>NN</td>
<td>Base</td>
<td>0.761</td>
<td>0.761</td>
<td>0.763</td>
<td>0.763</td>
<td>0.765</td>
<td>0.766</td>
<td>0.766</td>
<td>0.766</td>
<td>0.764 ± 0.002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pooling</td>
<td>0.761</td>
<td>0.762</td>
<td>0.764</td>
<td>0.763</td>
<td>0.766</td>
<td>0.767</td>
<td>0.768</td>
<td>0.768</td>
<td>0.765 ± 0.001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FATE</td>
<td><strong>0.770</strong></td>
<td><strong>0.769</strong></td>
<td><strong>0.771</strong></td>
<td><strong>0.772</strong></td>
<td><strong>0.773</strong></td>
<td><strong>0.774</strong></td>
<td><strong>0.774</strong></td>
<td><strong>0.772</strong></td>
<td><strong>0.772 ± 0.001</strong></td>
</tr>
<tr>
<td></td>
<td>DeepFM</td>
<td>Base</td>
<td>0.772</td>
<td>0.771</td>
<td>0.772</td>
<td>0.772</td>
<td>0.774</td>
<td>0.774</td>
<td>0.774</td>
<td>0.774</td>
<td>0.773 ± 0.001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pooling</td>
<td>0.772</td>
<td>0.772</td>
<td>0.773</td>
<td>0.774</td>
<td>0.776</td>
<td>0.776</td>
<td>0.776</td>
<td>0.776</td>
<td>0.774 ± 0.002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FATE</td>
<td><strong>0.781</strong></td>
<td><strong>0.780</strong></td>
<td><strong>0.782</strong></td>
<td><strong>0.782</strong></td>
<td><strong>0.784</strong></td>
<td><strong>0.784</strong></td>
<td><strong>0.784</strong></td>
<td><strong>0.784</strong></td>
<td><strong>0.783 ± 0.001</strong></td>
</tr>
</tbody>
</table>

Results: FATE achieves significantly improvements over Base/Pooling with different backbones (NN and DeepFM[1])

FATE can extrapolate for unseen features in dynamic data

Scalability Test for Large Datasets

**Results:** FATE yields linear time/space scalability w.r.t. data and feature sizes. **Promising for larger datasets and real systems**

The feature-data graph representation and GNN learning induces complexity $O(Bd)$.
Comparison with Other Learning Problems

- **Domain Adaption:**
  - Our differences: 1) same label distribution, 2) one task with different input feature space

- **Continual Learning:**
  - Our differences: 1) one-piece data, 2) no further re-training, 3) one task for learning

- **Open-Set Learning:**
  - Our differences: 1) same label space, 2) different input feature space

- **Zero-Shot Learning:**
  - Our differences: 1) no extra side information, 2) different feature space
Conclusions

Our contributions: **new problem setting** + **new method**

- Formulate the problem of open-world feature extrapolation
- Propose a graph-learning approach with new training techniques
- Provide theoretical insights on the generalization performance
- Empirically verify the effectiveness, applicality and scalability of new methods

Thanks for listening!