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

Anonymous Author(s)
Affiliation
Address
email

Abstract

We target open-world feature extrapolation problem where the feature space of input data goes through expansion and a model trained on partially observed features needs to handle new features in test data without further retraining. The problem is of much significance for dealing with features incrementally collected from different fields. To this end, we propose a new learning paradigm with graph representation and learning. Our framework contains two modules: 1) a backbone network (e.g., feedforward neural nets) as a lower model takes features as input and outputs predicted labels; 2) a graph neural network as an upper model learns to extrapolate embeddings for new features via message passing over a feature-data graph built from observed data. Based on our framework, we design two training strategies, a self-supervised approach and an inductive learning approach, to endow the model with extrapolation ability and alleviate feature-level over-fitting. We also provide theoretical analysis on the generalization error on test data with new features, which dissects the impact of training features and algorithms on generalization performance. Our experiments over several classification datasets and large-scale advertisement click prediction datasets demonstrate that our model can produce effective embeddings for unseen features and significantly outperforms baseline methods that adopt KNN and local aggregation.

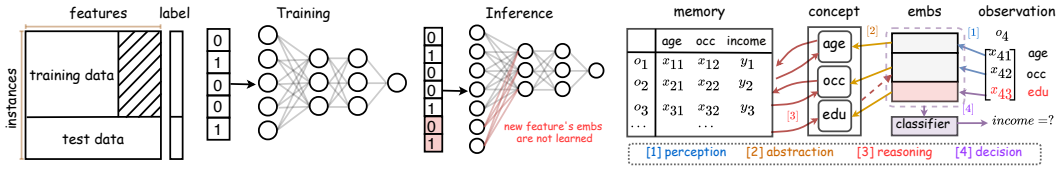
1 Introduction

Learning mapping from observation $\mathbf{x}$ (a vector of attribute features) to label y is a fundamental and pervasive problem in ML community, with extensive applications spanning from classification/regression tasks for tabular data to advertisement click prediction [18, 31, 10, 30, 21], item recommendation [20, 38, 11, 39], question answering [9, 8], or AI more broadly. Existing approaches focus on a fixed input feature space shared by training and test data. Nevertheless, practical ML systems interact with a dynamic open-world where features are incrementally collected. For example, in recommender/advertisement systems, there often occur new user profile features unseen before for current prediction tasks. Also, with the advances of multi-modal recognition [36] and federated learning [23], it is a requirement for a model trained with partial features to incorporate new features from other fields for decision-making on a target task.

A challenge stems from the fact that off-the-shelf neural network models cannot deal with new features without re-training on new data. As shown in Fig. 1(a), a neural network builds a mapping from input features to a hidden representation through a weight matrix in the first layer. Given new features as input, the network needs to be augmented with new weight parameters and cannot map the new features to a desirable position in latent space if without re-training.

However, model re-training would be tricky and bring up several issues. First, re-training a model with both previous and new features would be highly time-consuming and cannot meet the requirement

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.



(a) Limitation of neural network models for open-world feature extrapolation

(b) A new approach inspired by human's thinking

Figure 1: (a) The open-world feature extrapolation problem is defined over training data with partial features and test data with augmented feature space. Existing neural models cannot deal with new features without re-training. (b) Oue approach has an analogy to thinking process in human's brain. Imagine a person trained on using features *age*, *occ* to predict one's *income* and tested on test cases with a new feature *edu*. There are four processes from acquiring observation to giving final prediction.

for online systems. Alternatively, one can re-train a trained model only on new features, which will induce risks for over-fitting new data or forgetting previous data.

Different from machines, fortunately, humans are often equipped with the ability for extrapolating to unseen features and distill the knowledge in new information for solving a target task without any re-training on new data. The inherent gap between existing ML approaches and human intelligence raises an research question: *Can we design a ML model that is trained on one set of features and able to generalize to combine new unseen features for the same task without further training?*

To find desirable solutions to this non-trivial question is challenging. Let us resort to how humans think and act in physical world scenarios. Imagine that we are well trained on predicting a person's income with one's age and occupation from historical records. Now we are asked to estimate the income of a new person with his/her age, occ. and education (a new feature). We can modularize the process in brain systems that distill and leverage the knowledge in new observations into four steps, as shown in Fig. 1(b). 1) *Perception*: the observations are recognized by our senses; 2) *Abstraction:* the perceived information is aligned with concepts in our cognition; 3) *Reasoning:* we search similar concepts and observations in our memory to understand and absorb the new knowledge; 4) *Decision:* with new understanding and abstraction, we make final decisions for the prediction.

The above human's thinking process inspires the methodology in our paper where we propose a new learning paradigm for open-world feature extrapolation. Our proposed framework contains two modules: a backbone network, which can be a feedforward neural network, and a graph neural network. 1) The backbone network first maps input features to embeddings, which can be seen as *perception* from observation. 2) We then treat observed data matrix (each row represents a feature vector for one instance) as a feature-data bipartite graph, which explicitly define proximity and locality structures among features and instances. Then a graph neural network is harnessed for neural message passing over adjacent features and instances in a latent space (*abstraction*). 3) The GNN will inductively compute embeddings for new features based on those of existing ones, mimicking the *reasoning* process from familiar concepts to new ones in our brain. 4) The newly obtained embeddings that capture both semantics and feature-level relations will be used to obtain hidden representations of new data with unseen features and make final *decisions*.

To endow the model with ability for extrapolation to new features, we propose two training algorithms, one by self-supervised learning and one by inductive learning. The proposed model and its learning algorithm can easily scale to large-scale dataset (with millions of features and instances) via mini-batch training. We also provide a theoretical analysis on the generalization error on test data with new features, which allows us to dissect the impact of training features and algorithms on generalization performance. To verify our approach, we conduct extensive experiments on several real-world datasets, including six classification datasets from biology, engineering and social domains with diverse features, as well as two large-scale datasets for advertisement click prediction. Our model is trained on training data using partial features and tested on test data with a mixture of seen and unseen features. The results demonstrate that 1) our approach consistently outperform models not using new features for inference; 2) our approach achieves averagely 30.0% higher Accuracy than baseline methods using KNN, pooling or local aggregation for feature extrapolation; 3) our approach even significantly exceeds models using incremental training with new features.

We summarize our contributions as follows. 1) We formulate open-world feature extrapolation problem and show that it is feasible to extend neural models for extrapolating to new features without

re-training; 2) We propose a new graph-learning model and two training algorithms for feature extrapolation problem; 3) Our theoretical analysis shows that the generalization error for data with new features relies on the number of training features and the randomness in training algorithms; 4) We conduct comprehensive experiments and show the effectiveness, applicality and scalability of proposed method.

2 Methodology

We focus on attribute features as input in this paper. An input instance is a vector $\mathbf{r}_i = [r_{im}]_{m=1}^d \in \mathbb{R}^d$ where each entry r_{im} denotes a *raw feature* (like age, occ., edu., etc.). If r_{im} is a discrete/categorical raw feature with R_m possible values, its space is an integer set $\{0, 1, \cdots, R_m - 1\}$. If $r_{im'}$ is a continuous one, a common practice is to convert it into a discrete feature within space $\{0, 1, \cdots, R_{m'} - 1\}$ by evenly division [19] or log transformation [18]. We call R_m as *cardinality* for m-th raw feature.

An effective way to handle attribute features is via one-hot encoding [10, 19, 14]. For r_{im} with cardinality R_m we convert it into a R_m-dimensional one-hot vector $\mathbf{x}_i^m$ where the unique 1 indexes the value. In this way, one can convert an input $\mathbf{r}_i$ into a concatenation of one-hot vectors:

$$\mathbf{x}_i = [\mathbf{x}_i^1, \mathbf{x}_i^2, \cdots, \mathbf{x}_i^d], \text{ where } 1 \leq \forall m \leq d, \ \mathbf{x}_i^m \in \{0,1\}^{R_m} \text{ is a one-hot vector.} \tag{1}$$

Use x_{ij} to denote the j-th entry of $\mathbf{x}_i$ and we call each x_{ij} as *feature* in this paper. Assume $D = \sum_{m=1}^d R_m$ denotes the feature number.

We next give a formal definition for open-world feature extrapolation problem in this paper: given training data $\{(\mathbf{x}_i, y_i)\}_{i \in I_{tr}}$ where $\mathbf{x}_i \in \mathcal{X}_{tr} = \{0,1\}^D$, $y_i \in \mathcal{Y}$ and I_{tr} is a set of indices, we aim to learn a model that can generalize to test data $\{\overline{\mathbf{x}}_{i'}, y_{i'}\}_{i' \in I_{te}}$ where $\overline{\mathbf{x}}_{i'} \in \mathcal{X}_{te} = \{0,1\}^{\overline{D}}$, $y_{i'} \in \mathcal{Y}$ and I_{te} is another set of indices. We term $\mathcal{X}_{tr}$ as *training feature space* and $\mathcal{X}_{te}$ as *test feature space*. We assume 1) the label space $\mathcal{Y}$ is shared by training and test data, and 2) $\mathcal{X}_{tr} \subset \mathcal{X}_{te}$, i.e., test feature space is an extension of training feature space. The feature space expansion stems from two possible causes: 1) there appear new raw features incrementally collected from other fields (i.e., d increases) or 2) there appear new values out of the known space of existing raw features (i.e., R_m increases).

2.1 Proposed Model

Our model contains three parts: 1) feature representation that builds a bipartite feature-data graph from input data; 2) a backbone network which is essentially a neural network model that predicts the labels when fed with input data; 3) a GNN model that inductively compute features' embeddings based on their proximity and local structures to achieve feature extrapolation.

Feature Representation with Graphs. We stack the feature vectors of all the training data as a matrix $\mathbf{X}_{tr} \in \{0,1\}^{N \times D}$ where $N = |I_{tr}|$. We treat each feature and instance as nodes and construct a bipartite graph between them. Formally, we define the node set $F_{tr} \cup I_{tr}$ where $F_{tr} = \{f_j\}_{j=1}^D$ with f_j the j-th feature and $I_{tr} = \{o_i\}_{i=1}^N$ with o_i the i-th instance in training set. The binary matrix $\mathbf{X}_{tr}$ constitutes an adjacency matrix where the non-zero entries indicate edges connecting two nodes in F_{tr} and I_{tr}, respectively. The induced feature-data bipartite graph will play an important role in our extrapolation approach. The representation is flexible for variable-size feature set, enabling our model to handle test data $\overline{\mathbf{x}}_{i'} \in \{0,1\}^{\overline{D}}$ with $\mathbf{X}_{te} = [\overline{\mathbf{x}}_{i'}]_{i' \in I_{te}}$.

Backbone Networks. We next consider a prediction model $h_\theta(\cdot)$ as a backbone network that maps data features $\mathbf{x}_i$ to predicted label $\hat{y}_i$. Without loss of generality, a default choice for h_θ is a feedforward neural network. The first layer serves as an embedding layer which shrinks $\mathbf{x}_i$ into a H-dimensional hidden vector $\mathbf{z}_i = \mathbf{x}_i \mathbf{W}$ where $\mathbf{W} \in \mathbb{R}^{D \times H}$ denotes a weight matrix. The subsequent network (called *classifier*) is often a stack of neural layers that predicts label $\hat{y}_i = \text{FFN}(\mathbf{z}_i; \phi)$. We use the notation $\hat{y}_i = h(\mathbf{x}_i; \phi, \mathbf{W})$ to highlight two sets of parameters and $\theta = [\phi, \mathbf{W}]$.

Interestingly, the matrix multiplication in the embedding layer is equivalent to a two-step procedure: 1) a concatenation of indexed feature embeddings from a lookup table and 2) a permutation-invariant aggregation. To illustrate this equivalence, we consider $\mathbf{W}$ as a stack of weight vectors $\mathbf{W} = [\mathbf{w}_j]_{j=1}^D$ where $\mathbf{w}_j \in \mathbb{R}^{1 \times H}$ corresponds to the embedding of feature f_j. The non-zero entries in $\mathbf{x}_i$ will index the corresponding rows of $\mathbf{W}$ and induce a set of embeddings $\{\mathbf{z}_i^m\}_{m=1}^d$ where $\mathbf{z}_i^m$ is the embedding

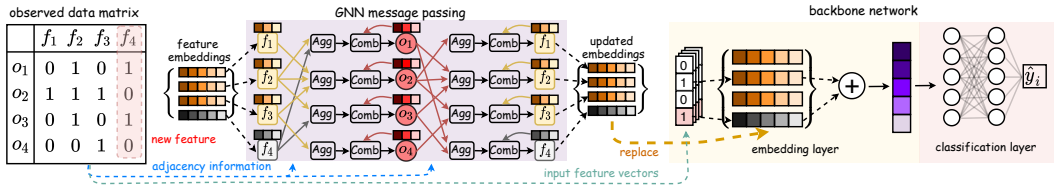


Figure 2: Feedforward computation of proposed model with input data $\{\mathbf{x}_i\}$ from which we build a feature-data graph between feature nodes $\{f_j\}$ and instance nodes $\{o_i\}$. A GNN model is harnessed to inductively compute features' embeddings via message passing and a backbone network leverages the updated embeddings to predict labels $\{y_i\}$.

given by x_i^m. After that, the hidden representation can be obtained by aggregation $\mathbf{z}_i = \sum_{m=1}^{d} \mathbf{z}_i^m$ which is permutation-invariant w.r.t. the order of feature embeddings in $\{\mathbf{z}_i^m\}$.

The permutation-invariant property opens a way for handling variable-length feature vectors $\overline{\mathbf{x}}_{i'}$. Essentially, on condition that we have embeddings for input features, we can add them up to get the fixed-dimensional hidden representation $\mathbf{z}_{i'}$ as input for the subsequent classifier. Therefore, the problem boils down to learning feature embeddings, especially *how to extrapolate for embeddings of new features based on those of existing ones*.

Remark. Instead of using sum aggregation, some existing architectures consider concatenation of d embedding vectors $\mathbf{z}_i^m$'s, which is essentially equivalent to sum aggregation (see Appendix A for more details). Therefore, the permutation-invariant property holds for widely adopted deep models.

GNN for Feature Extrapolation. We proceed to propose a graph neural networks (GNN) model for embedding learning with the feature-data graph. Our key insight is that the bipartite graph explicitly embodies features' co-occurrence in observed instances, which reflects the proximity among features. Once we conduct message passing for feature embeddings over the graph structures, the embeddings of similar features can be leveraged to compute and update each feature's embedding. The model can learn to extrapolate for new features' embeddings using those of existing features with locality structures in a data-driven manner. The message passing over the defined graph representation is *inductive* w.r.t. variable-sized feature nodes and instance nodes, which enables the model to tackle new feature space with distinct feature sizes and supports.

Specifically, we consider the embeddings $\mathbf{w}_j$ as an initial state $\mathbf{w}_j^{(0)}$ for node f_j in $\mathcal{F}_{tr}$. The initial states of instance nodes are set as zero vectors with equal dimension as the feature nodes, i.e. $\mathbf{s}_i^{(0)} = \mathbf{0}$. The interaction between two sets of nodes $\{\mathbf{w}_j\}_{j=1}^{D}$ and $\{\mathbf{s}_i\}_{i=1}^{N}$ can be modeled via graph neural networks where the node states in the l-th layer are updated by

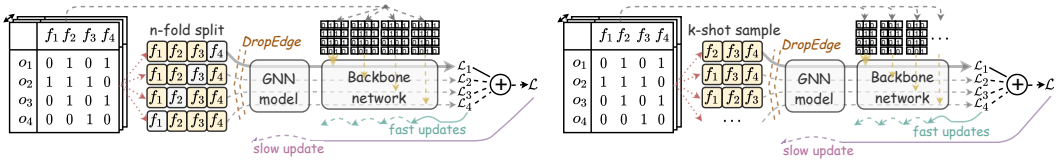
$$\mathbf{s}_i^{(l)} = \mathbf{P}^{(l)} \textsc{Comb}\left(\mathbf{s}_i^{(l-1)}, \textsc{Agg}(\{\mathbf{w}_k^{(l-1)}\} | \forall k, x_{ik} = 1)\right), \tag{2}$$

$$\mathbf{w}_j^{(l)} = \mathbf{P}^{(l)} \textsc{Comb}\left(\mathbf{w}_j^{(l-1)}, \textsc{Agg}(\{\mathbf{s}_k^{(l-1)}\} | \forall k, x_{jk} = 1)\right), \tag{3}$$

where $\mathbf{P}^{(l)} \in \mathbb{R}^{H \times H}$ is a weight matrix and we do not use non-linearity since it would degrade the performance empirically. For any new feature $f_{j'}$ in test data, we can set its initial state as a zero vector $\mathbf{w}_{j'}^{(0)} = \mathbf{0}$. The GNN model outputs updated embeddings for feature nodes and we further use them to replace the feature embeddings in the backbone network. Fig. 2 presents an illustration of the feedforward computation of proposed model. With L-layer GNN, we have $\hat{\mathbf{W}} = [\mathbf{w}_j^{(L)}]_{j=1}^{D} = g(\mathbf{W}, \mathbf{X}; \omega)$ and $\hat{y}_i = h(\mathbf{x}_i; \phi, \hat{\mathbf{W}})$, where $\hat{\mathbf{W}}$ denotes updated feature embeddings by GNN network g with parameter $\omega = \{\mathbf{P}^{(l)}\}_{l=1}^{L}$ and $\mathbf{X} = \mathbf{X}_{tr}$ for training or $\mathbf{X} = \mathbf{X}_{te}$ for test.

2.2 Model Learning

We next discuss approaches for model training. In order to enable the model to extrapolate for new features, we put forward two useful strategies. 1) *Feature-level data partition:* dividing the training data into several sets with different features as observed ones; 2) *Asynchronous updates:* decoupling the training of backbone network and GNN network and using different updating speeds for them in



(a) self-supervised learning (b) inductive supervised learning

Figure 3: Illustration of two proposed training methods: (a) Self-supervised learning with n-fold splitting ($n = 4$), and (b) Inductive learning with k-shot sampling ($k = 3$). The two methods differ in feature-level data partition, and both consider asynchronous updating rule: the backbone network is updated with n-folds proxy data before the GNN network is updated once with an accumulated loss.

a nested manner. The illustration is presented in Fig. 3. We proceed to propose two specific strategies for data partition.

Self-supervised Learning with N-fold Splits. To mimic new features in the future, we can mask some observed features and let the model use the remaining features to estimate the embeddings of the masked ones. For a given feature set $F_{tr} = \{f_j\}_{j=1}^{D}$ of data $\{\mathbf{x}_i\}_{i \in I_{tr}}$, we consider an n-fold splitting method: the features are first randomly shuffled and evenly divided into n disjoint subsets, denoted by $\{\overline{F}_s\}_{s=1}^{n}$. We then consider asynchronous updating rule for two networks: each iteration contains n times updates for backbone network and one update for GNN model. For the s-th update of the backbone, we mask features in $\overline{F}_s$ and set the initial states of masked features as zero vectors before fed into GNN. The GNN network will use the adjacency matrix $\mathbf{X}_{tr}$ to compute updated embeddings for the masked features. The embedding layer will be composed of updated embeddings of masked features and initial states of the remaining features, based on which the backbone network outputs prediction for each $\mathbf{x}_i$ and compute the loss (e.g., cross-entropy for classification). After n-step updates for backbone network, we use the accumulated loss to update GNN model. The training procedure will repeat the above process until a given time budget.

Inductive Learning with K-shot Samples. Alternatively, we can sample over the feature set and only expose partial features to the model for each update. For the s-th update, we randomly sample k observed features from F_{tr} as F_s and extract the corresponding columns of $\mathbf{X}_{tr} = [\mathbf{x}_i]_{i \in I_{tr}}$ to form a proxy data matrix $\mathbf{X}_s = [\mathbf{x}_i^s]_{i \in I_{tr}}$. Then $\mathbf{X}_s$ is fed into GNN to obtain updated embeddings of features in F_s, based on which the backbone network outputs prediction for each instance using $\mathbf{x}_i^s$.

By contrast, the n-fold splitting contributes to better training stability since the model is updated on each feature in each iteration, while the inductive learning adds more randomness which can help to enhance model's generalization. We will further discuss their pros and cons in our experiments.

DropEdge for Regularization. In order to further alleviate over-fitting on training features, we use the DropEdge [33] to regularize our model. We consider a threshold ρ and randomly set nonzero entries in $\mathbf{X}_{tr}$ (for self-supervised) or $\mathbf{X}_s$ (for inductive) as zero for each feedforward computation,

$$\tilde{\mathbf{X}}_{tr} = \textsc{DropEdge}(\mathbf{X}_{tr}, \rho) = \{x_{ij} | x_{ij} \in \mathbf{X}_{tr}, e_{ij} > \rho\}, \text{ where } e_{ij} \sim \mathcal{U}(0,1)]. \tag{4}$$

Scaling to Large Systems. To handle prohibitively large datasets for practical systems, we can divide data matrix into mini-batches along the instance dimension. Then, we feed each mini-batch into the model for once model training or inference. Note that the data partition is only necessary for the instance dimension (row) instead of the feature dimension (column) because of the sparse nonzero entries in $\mathbf{X}_{tr}$. Quantitatively, the number of nonzero features for each instance is less than d (a relatively small value), so the edge number in each mini-batch will be controlled within $O(Bd)$ (assume a mini-batch contains B instances). Hence, the space cost can be effectively controlled using instance-level mini-batch partition. We present the complete training algorithm in Appendix B.

3 Generalization Analysis

In this section, we analyze the generalization error on test data with new features. We focus our analysis on the case of inductive learning and leave the discussion for self-supervised learning as future work. Also, we simplify the settings for analysis: 1) the backbone network is a two-layer FFN (an embedding layer $\mathbf{W}$ plus a fully-connected layer Φ) with sigmoid output; 2) the GNN network is

a L-layer GCN which takes mean pooling aggregation over neighbored nodes' and centered node's features without linear transformation in each layer; 3) the training algorithm is SGD. With above settings, the model can be written as $\hat{y}_i = h(\mathbf{x}_i; \phi, g(\mathbf{W}, \mathbf{X}; \omega)) = \sum_{i' \in \mathcal{N}_{\tilde{L}}(i) \cup \{i\}} c_{ii'} \mathbf{x}_{i'} \mathbf{W} \Phi$ where $\mathcal{N}_{\tilde{L}}(i)$ contains all the $\mathbf{x}_{i'}$'s that appear in the $\tilde{L}$-hop neighbors of $\mathbf{x}_i$ in the feature-data graph, $\tilde{L} = 2 \cdot \lfloor \frac{L}{2} \rfloor$, and $c_{ii'}$ is the connectivity between $\mathbf{x}_i$ and $\mathbf{x}_{i'}$ through L-layer graph convolution. More details for the derivation are in Appendix C.

The data generation process can be described as follows. First, a feature set $F_{tr} = \{f_j\}_{j=1}^{D}$ is sampled from an unknown distribution $\mathcal{F}_{tr}$. Then data $\{(\mathbf{x}_i, y_i)\}_{i \in I_{tr}}$ are sampled from a distribution $\mathcal{D}_{tr}$ whose support is over $\mathcal{X}_{F_{tr}} \times \mathcal{Y}$ and $\mathbf{X}_{tr} = \{\mathbf{x}_i\}_{i \in I_{tr}}$ and $Y_{tr} = \{y_i\}_{i \in I_{tr}}$. Using $\psi = [\theta, \omega]$, the model can be denoted by $\hat{Y} = h(\mathbf{X}; \psi)$ (with a simplification from $\hat{Y} = \{\hat{y}_i\}_{i \in I} = \{h(\mathbf{x}_i; \psi)\}_{i \in I}$) with loss function $\mathcal{L}(\hat{Y}, Y) = \mathcal{L}(\hat{Y}, h(\mathbf{X}; \omega)) = \frac{1}{N} \sum_{i \in I} l(y_i, \hat{y}_i)$ where $N = |I|$.

In training stage, we randomly sample over F_{tr} to obtain a series of feature subsets $\{F_s\}$ and randomly partition the training instances into mini-batches $\{(\mathbf{X}_{F_s}, Y_{F_s})\}$. The procedure generates a sequence of data sub-matrices (with partially observed features and instances) that are sequentially fed into the model and compute the updating gradients. With a given training data $(\mathbf{X}_{tr}, Y_{tr})$, we define $\mathcal{S}$ as a set of all the data sub-matrices that could be sampled during the training

$$\mathcal{S} = \{(\mathbf{X}_{F_1}, Y_{F_1}), (\mathbf{X}_{F_2}, Y_{F_2}), \cdots, (\mathbf{X}_{F_m}, Y_{F_m}), \cdots, (\mathbf{X}_{F_M}, Y_{F_M})\}.$$

The training process can be seen as a sequence of operations each of which samples an observed data from $\mathcal{S}$ in an i.i.d. manner and computes gradients for one SGD update. Define $A_{\mathcal{S}}$ as a learning algorithm trained on $\mathcal{S}$, which gives trained model $h(\mathbf{X}; \psi_{\mathcal{S}})$ simplified as $f_{\mathcal{S}}$. The generalization error $R(h_{\mathcal{S}})$ can be defined as

$$R(h_{\mathcal{S}}) = \mathbb{E}_{F \sim \mathcal{F}, (\mathbf{X}_F, Y_F) \sim \mathcal{D}_F}[\mathcal{L}(Y_F, h(\mathbf{X}_F; \psi_{\mathcal{S}}))]. \tag{5}$$

The empirical risk that our approach optimizes with the training data would be

$$R_{emp}(h_{\mathcal{S}}) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}(Y_{F_m}, h(\mathbf{X}_{F_m}; \psi_{\mathcal{S}})). \tag{6}$$

We study the expected generalization gap

$$\mathbb{E}_A[R(h_{\mathcal{S}}) - R_{emp}(h_{\mathcal{S}})], \tag{7}$$

where the expectation is taken over the randomness of $A_{\mathcal{S}}$ stemming from sampling for SGD updates.

We assume that the loss function $l(y_i, \hat{y}_i)$ is Lipschitz-continuous and smooth w.r.t. the model output $\hat{y}_i$. Concretely, we have 1) $|l(y, f(\cdot)) - l(y, f'(\cdot))| \leq \beta |f(\cdot) - f'(\cdot)|$ and 2) $|\nabla l(y, f(\cdot)) - \nabla l(y, f'(\cdot))| \leq \beta' |\nabla f(\cdot) - \nabla f'(\cdot)|$. Such condition can be satisfied by widely used loss functions such as cross-entropy and MSE. Then we have the following result (see Appendix C for proof).

Theorem 1. *Assume the loss function is bounded by $l(y_i, \hat{y}_i) \leq \lambda$. For a learning algorithm trained on data $\{\mathbf{X}_{tr}, Y_{tr}\}$ with T iterations of SGD updates, with probability at least $1 - \delta$, the expected generalization gap would be bounded by*

$$\mathcal{O}(\frac{d^T}{M}) + \left(\mathcal{O}(\frac{d^T}{M^2}) + \lambda\right) \sqrt{\frac{\log(1/\delta)}{2M}}. \tag{8}$$

The theorem shows that the generalization error depends on the number of raw features in training data and the size of set $\mathcal{S}$. Notice that the latter is determined by sampling strategies for partitioning the training data into sub-matrices. If the sampling introduces more randomness, $\mathcal{S}$ would become larger, contributing to better generalization ability. However, a large $\mathcal{S}$ would also lead to large variance for training and further cause undesirable optimization error. Therefore, there exists an interesting trade-off w.r.t. how to sample/partition observed features in training stage. Furthermore, the generalization gap also depends on d, and more training features would result in looser bound. This is because a larger d would require the model to deal with more features. As the network becomes *wider* and complex, it would be more prone for over-fitting.

4 Experiments

We apply our model FATE (for FeATure Extrapolation Networks) to several real-world datasets to verify its effectiveness. First, we consider six classification datasets from UCI Machine Learning Repository [1]: Gene, Protein, Robot, Drive, Calls and Github. These datasets are collected from domains like biology, engineering and social networks. The feature numbers are ranged from 219 to 4006 and the instance numbers vary from 1080 to 58509. Furthermore, we consider two large-scale datasets Avazu and Criteo from real-world online advertisement system whose goal is to predict the Click-Through Rate (CTR) of exposed advertisement to users. The two datasets have ~ 40 million clicking/non-clicking records as instances and ~ 2 million features. More dataset information and implementation details are in Appendix D and E, respectively.

4.1 Experiment on UCI Datasets

Setup. We randomly split all the instances into training/validation/test data with the ratio 6:2:2. Then we randomly select a certain ratio ($30\% \sim 80\%$) of features as *observed* ones and use the remaining as *unobserved* ones. The model is trained with the *observed* features of training instances and tested with *all* the features of testing data. We adopt Accuracy as metric for datasets with more than two classes (Gene, Protein, Robot, Drive and Calls) and ROC-AUC for Github with two classes.

Implementation. We specify FATE in the following ways. 1) Backbone: a three-layer feedforward NN. 2) GNN: a four-layer GCN. 3) Training: self-supervised learning with n-fold splitting. Several baselines are considered for comparison and their architectures are all specified as a three-layer feedforward NN. First, *Base-NN*, *Average-NN*, *Pooling-NN* and *KNN-NN* are all trained on training instances with observed features. Then *Base-NN* only uses test instances' observed features for inference. *Average-NN* uses averaged embeddings of observed features as those of unobserved ones. *Pooling-NN* (resp. *KNN-NN*) compute embeddings for unobserved features via replacing our GNN with non-parametric mean pooling aggregation over neighborhoods (resp. averaging embeddings of observed features with top 20% Jaccard similairties). Furthermore, we consider *Oracle-NN* using all the features of training data for training and *INL-NN* that is first trained on training data with observed features and then re-trained on training data with the remaining features.

Results and Discussions. We report the mean Accuracy/ROC-AUC in Fig. 4 with different ratios of observed features ranging from 0.3 to 0.8. As we can see, our model FATE achieves the best results throughout all the cases in six datasets. Concretely, FATE achieves averagely 7.3% higher Accuracy and 1.3% higher ROC-AUC over Base-NN which uses partial features for inference. The improvements are statistically significant under 95% confidence level. The results show that FATE can learn effective embeddings for new features that contribute to better performance for classification task. Furthermore, FATE achieves averagely 29.8% higher Accuracy and 0.5% higher ROC-AUC over baselines Average-NN, KNN-NN and Pooling-NN. These baseline methods perform worse than Base-NN especially when observed features are few, which demonstrates that directly aggregating embeddings of observed features for extrapolation would degrade the performance. By contrast, FATE possesses superior capability for extrapolating to new unseen features. Even with 30% observed features the model is able to distill the useful knowledge from 70% unobserved features without re-training, providing decent classification performance. Notably, compared with INL-NN, FATE even achieves higher accuracy in most cases with a 4.1% Accuracy improvement on average. The reason is that INL-NN is prone for over-fitting on new data and forgetting the previous one. Finally, FATE achieves very close performance to Oracle-NN with sufficient observed features and can even slightly exceeds it with 80% features in Gene, Robot and Github. In fact, the GNN network in FATE can not only achieve feature extrapolation, but also capture hidden relations among features, which can be another merit of our method.

4.2 Experiment on CTR Prediction

Setup. We consider data splitting in chronological order to simulate real-world cases. For Avazu dataset which contains time information in ten days, we use the data of first/second day for training/validation and the data of the third to tenth days for test. For Criteo dataset whose records are ranked in temporal order, we split the dataset into ten continual sets with equal size and use the first/second set for training/validation and the third to tenth sets for test. With such data splitting, we can naturally obtain test data with new features not appeared in the training data. Overall, the

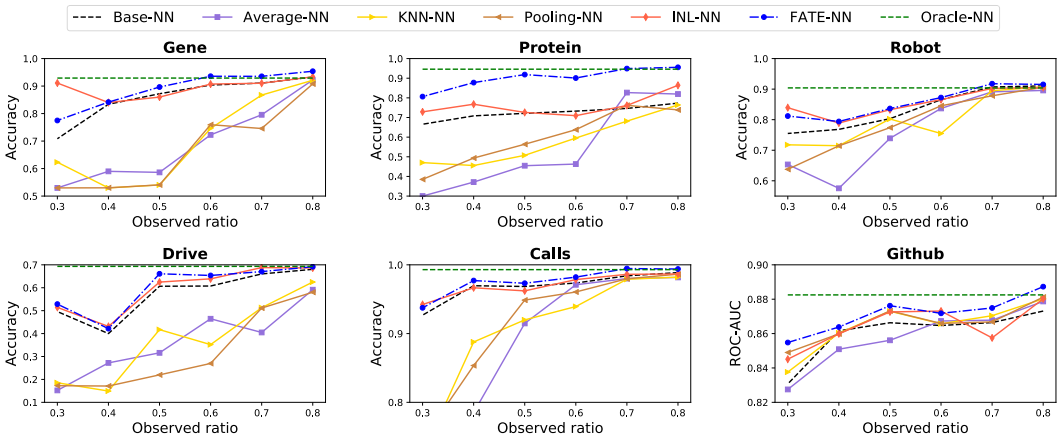


Figure 4: Accuracy/ROC-AUC results for UCI datasets with $30\% \sim 80\%$ observed features for training. We run each experiment five times with different random seeds and report averaged scores.

Table 1: ROC-AUC results for 8 test splits (T1-T8) on Avazu and Criteo datasets.

Dataset	Backbone	Model	T1	T2	T3	T4	T5	T6	T7	T8	Overall
Avazu	DNN	Base	0.666	0.680	0.691	0.694	0.699	0.703	0.705	0.705	0.693 ± 0.012
		Pooling	0.6551	0.671	0.6834	0.6839	0.6891	0.6942	0.6978	0.697	0.684 ± 0.011
		FATE	**0.689**	**0.699**	**0.708**	**0.710**	**0.715**	**0.720**	**0.721**	**0.721**	$\mathbf{0.710} \pm 0.010$
	DeepFM	Base	0.675	0.684	0.691	0.693	0.697	0.701	0.703	0.703	0.693 ± 0.009
		Pooling	0.666	0.676	0.685	0.685	0.688	0.693	0.694	0.694	0.685 ± 0.009
		FATE	**0.692**	**0.702**	**0.711**	**0.714**	**0.718**	**0.722**	**0.724**	**0.724**	$\mathbf{0.713} \pm 0.010$
Criteo	DNN	Base	0.761	0.761	0.763	0.763	0.765	0.766	0.766	0.766	0.764 ± 0.002
		Pooling	0.761	0.762	0.764	0.763	0.766	0.767	0.768	0.768	0.765 ± 0.001
		FATE	**0.770**	**0.769**	**0.771**	**0.772**	**0.773**	**0.774**	**0.774**	**0.774**	$\mathbf{0.772} \pm 0.001$
	DeepFM	Base	0.772	0.771	0.772	0.772	0.774	0.774	0.774	0.774	$0.773 + 0.001$
		Pooling	0.772	0.772	0.773	0.774	0.776	0.776	0.776	0.776	0.774 ± 0.002
		FATE	**0.781**	**0.780**	**0.782**	**0.782**	**0.784**	**0.784**	**0.784**	**0.784**	$\mathbf{0.783} + 0.001$

ratio between observed features in training set and new features in each test set is nearly 1:1. We use ROC-AUC for evaluation.

Implementation Details. We consider two specifications for our backbone: 1) a three-layer feed-foward NN; 2) DeepFM, a widely used model for advertisement click prediction considering inter-feature interactions over NN. The GNN is a two-layer GraphSAGE model and training method is inductive learning with k-shot sampling. Also, we compare with baselines *Base* and *Pooling*. The *KNN* method would suffer from scalability issue in the two large datasets.

Results and Discussions. Table 1 reports the ROC-AUC results for different test splits, which show that FATE significantly outperforms Partial and Pooling in all the test splits using NN and DeepFM as backnones. Overall, compared with Base, FATE improves the ROC-AUC by 0.017/0.02 (resp. 0.008/0.01) with DNN/DeepFM as the backbone on Avazu (resp. Criteo). Note that even an improvement of 0.002 for ROC-AUC is considered significant in click prediction tasks [10, 14]. The results show that FATE can combine useful information in new features collected in the future for the target task without further training and has promising power for enhancing the real-world systems interacting with open world. Compared with Pooling, FATE achieves significant AUC improvements on two datasets. The reason is that directly using average pooling to replace the GNN convolution would lead to limited capacity and weakens its ability for effective concept abstraction and reasoning.

4.3 Further Discussions

Scalability. We study model's scalability w.r.t. different batch sizes B and number of features D in Fig. 6 and 7 which show that our training and inference time/space scale linearly on Criteo dataset.

Ablation Studies. We also provide thorough ablation studies in Table 3. The results show that 1) the DropEdge operation can indeed help regularize the training and bring up higher test accuracy; 2)

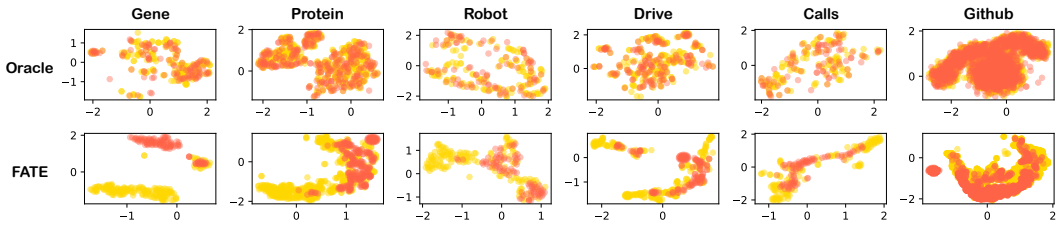


Figure 5: Visualization for t-SNE embeddings of FATE-NN's and Oracle-NN's produced feature embeddings. We mark observed features with red and the remaining ones with blue nodes.

using asynchronous updates for two networks can indeed lead to performance gain over end-to-end training; 3) the n-fold splitting and k-shot sampling will outperform each other in different cases and both exceeds leave-one-out partition. Table 4 compares using different sampling size k for sampling in inductive learning on Avazu and Criteo, which validates our theoretical analysis in Section 3. More discussions are in Appendix F.

Visualization. Fig. 5 presents a visualization of learned features' embeddings by FATE-NN and Oracle-NN. The results unveil two interesting insights that can interpret why FATE can sometimes outperform Oracle that uses all the features for training. First, FATE-NN's produced embeddings for observed and unobserved features possess distinct distributions in latent space, while the embeddings of Oracle-NN appear to have the same distribution. Such phenomenon implies that FATE manages to learn informative knowledge from new features. Second, the learned embeddings of FATE-NN form some particular structures (clusters, lines or curves) rather than uniformly distribute over the 2-D plane like Oracle-NN. It implies that FATE manages to learn hidden relations among features which could be helpful for downstream classification.

5 Links to Existing Works

We discuss the relationships of our problem setting, open-world feature extrapolation (OFE), and our model FATE with domain adaption (DA), continual learning (CL) and open-set learning (OSL).

Domain Adaption adapts a model trained on source domain to target with different distribution [4, 7, 25, 12, 13, 41, 24]. Our problem OFE is different from DA in two aspects: 1) the label space/distribution of training and test data is the same for OFE, while DA often mostly considers different label distributions for source and target domains; 2) OFE focus on combining new features that are related to the current task, while DA considers different tasks from different domains.

Continual Learning, or lifelong learning, aims at enabling a single model to learn from a stream of data from different tasks that cannot be seen at one time [32, 34, 28, 27]. By contrast, there are two-fold differences of OFE. First, OFE does not allow finetuning or further retraining on new data, which can be more challenging than CL. Second, CL mostly assume each piece of data in the stream is from different tasks with different labels to handle. The key challenge of CL is the catastrophic forgetting [32, 27] that requires the model to balance a trade-off between previous and new tasks, while FATE is free from such issue in nature since we do not require incremental learning.

Open-set Learning is another line of researches that relate with us. Differently, open-set recognition mostly focus on expansion of label sets [5, 6, 15, 40, 26, 35]. To our knowledge, we are the first to study feature set expansion and also the first to solve open-set learning via graph learning.

Extension and Outlooks. Our works can be extended to solve more problems and push the development in broader areas. 1) The input feature vector can be replaced by feature maps given by CNN or word/sentence embeddings by Transformer [37], for handling multi-modal data in the context of federated learning [23] or multi-view learning [36, 2, 3]; 2) as shown in experiments, the classification layers can be replaced by more complex models with feature-level interactions [19, 22, 16, 21] or advanced architectures for complicated decision and reasoning tasks [29, 17]; 3) our graph representation and GNN model can also be extended to directly handle continuous features by treating feature values as edge weights. We leave these interesting directions for future works.

6 Conclusions

We have presented a new framework to address new features unseen in training, by formulating it as the open-world feature extrapolation problem. We propose a learning paradigm via inductive graph learning and harness a graph neural network as an upper model to conduct message passing over a feature-data graph. The message passing can compute embeddings for new features with those of existing ones, mimicking the abstraction and reasoning process in human's brain. We also put forward two training strategies for more effective feature extrapolation. Our theoretical results show that the generalization error depends on training features and learning algorithms. Experiments show the effectiveness of our model as well as its scalability to large-scale system.

References

[1] A. Asuncion and D. Newman. Uci machine learning repository. 2007.

[2] B. B. Avants, N. J. Tustison, and J. R. Stone. Similarity-driven multi-view embeddings from high-dimensional biomedical data. *Nature Computational Science*, 1:143–152, 2021.

[3] D. Baptista, P. G. Ferreira, and M. Rocha. Deep learning for drug response prediction in cancer. *Briefings Bioinform*, 22(1):360–379, 2021.

[4] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

[5] A. Bendale and T. E. Boult. Towards open world recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 1893–1902, 2015.

[6] A. Bendale and T. E. Boult. Towards open set deep networks. In *Conference on Computer Vision and Pattern Recognition*, pages 1563–1572, 2016.

[7] J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Wortman. Learning bounds for domain adaptation. In *Advances in Neural Information Processing Systems*, pages 129–136, 2007.

[8] W. Chen, M. Chang, E. Schlinger, W. Y. Wang, and W. W. Cohen. Open question answering over tables and text. In *International Conference on Learning Representations*, 2021.

[9] W. Chen, H. Zha, Z. Chen, W. Xiong, H. Wang, and W. Y. Wang. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In *Conference on Empirical Methods in Natural Language Processing*, pages 1026–1036, 2020.

[10] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. In *Workshop on Deep Learning for Recommender Systems*, pages 7–10, 2016.

[11] P. Covington, J. Adams, and E. Sargin. Deep neural networks for youtube recommendations. In *Conference on Recommender Systems*, pages 191–198, 2016.

[12] Y. Ganin and V. S. Lempitsky. Unsupervised domain adaptation by backpropagation. In F. R. Bach and D. M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1180–1189. JMLR.org, 2015.

[13] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. S. Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17:59:1–59:35, 2016.

[14] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He. Deepfm: A factorization-machine based neural network for CTR prediction. In *International Joint Conference on Artificial Intelligence*, pages 1725–1731, 2017.

[15] M. Hassen and P. K. Chan. Learning a neural-network-based representation for open set recognition. In *International Conference on Data Mining*, pages 154–162, 2020.

[16] X. He and T. Chua. Neural factorization machines for sparse predictive analytics. In *International Conference on Research and Development in Information Retrieval*, pages 355–364, 2017.

[17] D. A. Hudson and C. D. Manning. Learning by abstraction: The neural state machine. In *Advances in Neural Information Processing Systems*, pages 5901–5914, 2019.

[18] Y. Juan, Y. Zhuang, W. Chin, and C. Lin. Field-aware factorization machines for CTR prediction. In *Conference on Recommender Systems*, pages 43–50, 2016.

[19] Y. Juan, Y. Zhuang, W. Chin, and C. Lin. Field-aware factorization machines for CTR prediction. In *Conference on Recommender Systems*, pages 43–50, 2016.

[20] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[21] Z. Li, J. Zhang, Y. Gong, Y. Yao, and Q. Wu. Field-wise learning for multi-eld categorical data. In *Advances in Neural Information Processing Systems*, pages 6639–6649, 2020.

[22] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *International Conference on Knowledge Discovery & Data Mining*, pages 1754–1763, 2018.

[23] P. P. Liang, T. Liu, Z. Liu, R. Salakhutdinov, and L. Morency. Think locally, act globally: Federated learning with local and global representations. *CoRR*, abs/2001.01523, 2020.

[24] M. Long, Z. Cao, J. Wang, and M. I. Jordan. Conditional adversarial domain adaptation. In *Advances in Neural Information Processing Systems*, pages 1647–1657, 2018.

[25] Y. Mansour, M. Mohri, and A. Rostamizadeh. Domain adaptation: Learning bounds and algorithms. In *Conference on Learning Theory*, 2009.

[26] P. Oza and V. M. Patel. C2AE: class conditioned auto-encoder for open-set recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 2307–2316. Computer Vision Foundation / IEEE, 2019.

[27] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[28] G. I. Parisi, J. Tani, C. Weber, and S. Wermter. Lifelong learning of human actions with deep neural network self-organization. *Neural Networks*, 96:137–149, 2017.

[29] H. Ren, W. Hu, and J. Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *International Conference on Learning Representations*, 2020.

[30] S. Rendle. Factorization machines. In *International Conference on Data Mining*, pages 995–1000, 2010.

[31] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: estimating the click-through rate for new ads. In *The Web Conference*, pages 521–530, 2007.

[32] A. V. Robins. Catastrophic forgetting in neural networks: the role of rehearsal mechanisms. In *First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, ANNES '93, Dunedin, New Zealand, November 24-26, 1993*, pages 65–68. IEEE, 1993.

[33] Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020.

[34] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[35] L. Shu, H. Xu, and B. Liu. DOC: deep open classification of text documents. In *Conference on Empirical Methods in Natural Language Processing*, pages 2911–2916, 2017.

[36] Y. H. Tsai, P. P. Liang, A. Zadeh, L. Morency, and R. Salakhutdinov. Learning factorized multimodal representations. In *International Conference on Learning Representations*. OpenReview.net, 2019.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[38] M. Xu, R. Jin, and Z. Zhou. Speedup matrix completion with side information: Application to multi-label learning. In *Advances in Neural Information Processing Systems*, pages 2301–2309, 2013.

[39] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.