AdvPL: Adversarial Personalized Learning

Wei Du
University of Arkansas
wd005@uark.edu

Xintao Wu
University of Arkansas
xintaowu@uark.edu

***Abstract*—The data generation sources are increasing in the past few years, such as mobile devices, embedded sensors, various intelligent equipment and so forth. These increasing data sources push the deployment of deep learning models in a distributed manner. However, the traditional distributed deep learning is to build a global model over all collected data and may overlook specific components which are of vital importance to personalized users. In this paper, we propose a learning framework that allows an individual user to build a personalized model. Our framework consists of two stages, including efficient similar data selection from other users and adversarial training. Instead of selecting similar data by computing hand-designed similarity metrics, we train an auto-encoder and a GAN on individual user's data, and use them to request similar data from other users. To further improve the personalized model performance, we apply adversarial training to minimize the distribution discrepancy between requested data and user's own data. Experimental results demonstrate the effectiveness of the proposed framework.**

***Index Terms*—deep learning, personalized model, adversarial learning**

I. Introduction

The past few years have witnessed an increasing role that deep learning plays in various kinds of applications, such as image classification [1], text generation [2], recommendation systems [3] and other artificial intelligence (AI)-related tasks. More recently, data for training deep learning models is increasingly distributed among different users. A traditional approach for the deployment of deep learning model is to collect all these data into a central server and build a global model. An alternative way is to collaboratively learn a global model among distributed users via parameters exchange [4]. The key of previous works is to build a one-fit-all global model and use it to perform classification or prediction for all users. Although a global model captures generic information of training data from all users, it cannot fulfill the personalized demand for each individual user due to the diverse data distribution among different users.

It is necessary to construct a personalized model for each individual user to improve intelligent services. Although the global model learns generic information over all users, the specific features of each individual user's data, which are of vital importance to build the personalized data, are often overlooked by the global model. In many circumstances, building a personalized model can achieve better performance for specific users. For instance, personalized recommendation systems play important roles for many online services [5], such as production advertisement, sales promotion, and information recommendation. In addition, electronic health records [6] are used to predict disease progression and provide treatment plans. Since electronic health records are patient-specific, personalized models are more effective for individual users.

In fact, in many cases, an individual user can only collect a small amount of data, which is insufficient to build an accurate personalized model to achieve satisfactory performance. To improve the performance of personalized model, an individual user needs to request similar or complementary data from other users. Pioneer works of building personalized model have been done in medical field, e.g., Cheng et al. [6] develop a framework of collecting data from similar patients and then training a personalized model on the combination of these collected data and his own data.

However, there are two challenges in the pipeline of building personalized models. The first challenge is how to efficiently and effectively collect similar data from other users. It is obvious that randomly selecting data from other users may not improve the performance of personalized model as those collected data may be different from the user's own data. Previous research [6] applies similarity metrics to construct similarity index for data selection. However, it is burdensome to construct similarity index and the similarity metrics are often based on manually designed features. The computational cost is high when one compares paired data one by one due to the high operation complexity of $\mathcal{O}(n^2)$ especially when the number of data n is large. Moreover, it is often unclear to choose one appropriate similarity metric so most appropriate data can be collected. Another challenge is how to better train the personalized model with the requested data. Previous research simply combines the collected data with user's own data, which may not build the personalized model with high accuracy. This is because there may exist distribution discrepancy between the collected data and user's own data, especially when the user can only get a small amount of data from other users (e.g., due to privacy concerns or high data collection cost). Hence it is imperative to develop personalized learning models that can handle the potential distribution discrepancy.

Motivated by the above two challenges, in this paper, we develop a learning framework that enables an individual user to effectively collect data and build a robust model on the combination of the collected data and his own data. For data collection, we propose an approach of using an auto-encoder and a generative adversarial network (GAN) [7]. The auto-encoder is used to obtain data representation that is further

used to train the GAN. The trained encoder and the discriminator from GAN are sent to his neighbors. Each neighbor user uses the encoder to obtain the representation of his data and then uses the discriminator to calculate the probability score of his data. Data with high probability scores are combined with the user's original data to train the personalized model. The advantage of using GAN is that it can capture inherent properties of the underlying data without manually specifying features. To handle the distribution discrepancy in learning personalized models, we apply adversarial training. The core idea is to map both the requested data and the user's own data into the same feature space where the distribution discrepancy is minimized. Our adversarial training is analogous to the discriminator of the GAN. The role of the discriminator is to predict whether the generated features are from user's own data or the requested data.

The main contributions of this paper are summarized as follows. First, we demonstrate that building a global model is not an optimal choice for personalized prediction. Second, we develop a strategy based on auto-encoder and GAN to effectively collect similar data from other users. Third, instead of directly using the requested data for training, we apply adversarial training to minimize the distribution discrepancy between user's own data and the requested data to improve accuracy of the personalized model. Finally, we conduct extensive experiments and the results demonstrate the efficiency of the proposed framework.

II. Related Work

Distributed deep learning: Distributed deep learning from multiple sources has been long investigated in the past few years. In the pioneer work [4], the dataset is distributed in different machines and a global model is learned by exchanging parameters between participating users. Following this work, later researchers focus on how to train distributed deep learning models more efficiently. For instance, Chilimbi et al. [8] propose an efficient and scalable system to allow the training of distributed deep learning. Wen et al. [9] develop a strategy to reduce communication cost in distributed deep learning to accelerate the training process. Moreover, how to improve the performance of distributed deep learning has been investigated extensively [10]–[12]. Besides distributed deep learning in the data center setting, federated learning [13] is proposed to push deep learning to mobile and edge devices, in which mobile devices only have limited data and users are willing to collaboratively learn a joint model. Recent works focus on achieving security guarantee [14] and solving data statistical challenges [15]. However, the focus of the existing works is to learn a global model from all training data.

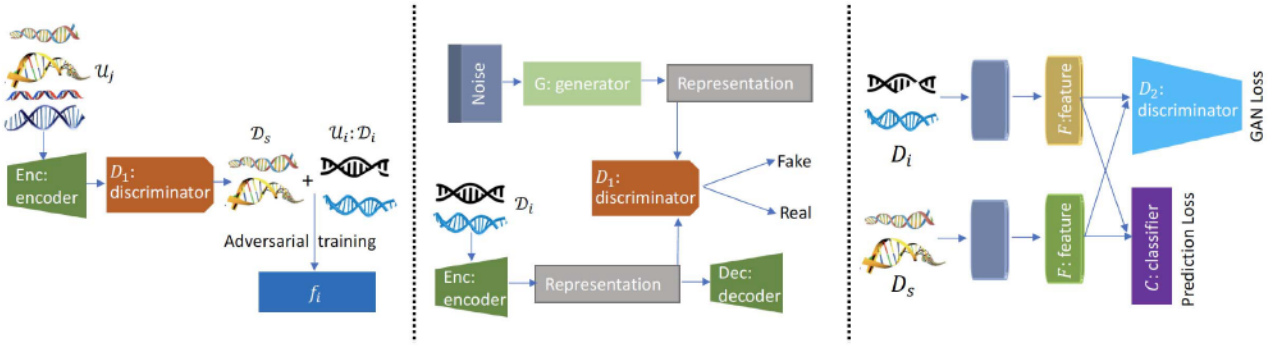
Personalized deep learning: Personalized model is popular in the areas of medicine and recommendation systems. In these areas, building a personalized model for each individual user is necessary since medical data and recommendation service are user-specific and a global model cannot capture personalized features. Similarity learning is fundamental for building a personalized patient model. In [16], Che et al. propose a dynamic temporal matching approach to find similar data for individual users and build a personalized RNN model for each individual user to predict disease. The authors [17] develop a CNN-based similarity method for paired data comparison and perform personalized disease prediction. Other works using different metrics for similarity personalized learning [18]–[20] are also reported. For personalized recommendation, one of the most popular approaches is based on collaborative filtering [5]. Similar to the personalized model in medicine, collaborative filtering predicts user preference by finding similar users who have similar taste and makes decisions. Various metrics are also used to measure the similarity between different users. For instance, Luo et al. [21] apply cosine similarity to build recommendation system for smart grid end users and Kouki et al. [22] use Pearson's correlation to compute data similarity. However, previous work based on similarity learning focus on tedious paired data comparison. Our work propose an efficient similarity approach based on auto-encoder and GAN and apply adversarial training to reduce potential distribution discrepancy.

Representation learning: Representation learning has been a well studied research area in the past few years [23], especially in computer vision, natural language processing and transfer learning. There are several works that apply representation learning to solve distribution discrepancy between different parties. For example, Tzeng et al. propose to learn domain invariant representations from the source domain and transfer to the target domain for prediction [24]. Liu et al. develop a framework that disentangles domain-invariant and domain-specific features in image translation and manipulation [25]. In [26], Gupta et al. extract invariant features between different agents in the reinforcement learning. Misra et al. propose a multi-task model to learn common representations between different tasks and use it to improve prediction performance [27]. Our work falls into the general area that exploits feature representation among different groups for performance improvement [28], [29]. Different from previous works, our work applies adversarial training to reduce the distribution discrepancy and learns the prediction task simultaneously to improve its performance.

III. ADVPL: Adversarial Personalized Learning

A. Framework Overview

The framework of our proposed AdvPL is illustrated in Figure 1. Consider there exist N individual users in this setting and each user with local dataset available. Let $\mathcal{U}$ and $\mathcal{D}$ represent the set of users and datasets, respectively, where each user $\mathcal{U}_i$ is associated with dataset $\mathcal{D}_i$. Suppose $\mathcal{D}_i$ contains M_i data samples, namely $(X_1, Y_1), (X_2, Y_2)\ , \ldots, (X_{M_i}, Y_{M_i})$. The goal of user $\mathcal{U}_i$ is to build a personalized classification model f_i. The classification model f_i takes $\mathcal{D}_i$ as input and minimizes the prediction error $Err_i = \sum_{m=1}^{M_i} L(f(X_m), Y_m)$. To boost the performance of his classifier f_i and reduce prediction error Err_i, $\mathcal{U}_i$ will request similar data from his neighbors. The neighbors include a set of users $\mathcal{U}_j$ where $j \in [1, N], j \neq i$.



(a) The framework of personalized learning (b) The training architecture of auto-encoder and GAN (c) The framework of adversarial training

Fig. 1: An overview of adversarial personalized learning

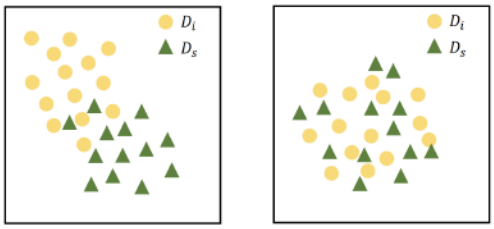


Fig. 2: (a) It depicts raw data distribution using dots and triangles between $\mathcal{D}_i$ and $\mathcal{D}_s$. In raw data space, the distribution between $\mathcal{D}_i$ and $\mathcal{D}_s$ is partly overlapped. (b) By optimizing a loss function that simultaneously maximizes overlap in the feature representation space and improves the model performance, we can reduce the distribution discrepancy between $\mathcal{D}_s$ and $\mathcal{D}_i$.

The workflow is shown in Algorithm 1. Lines 4 - 5 show the training process of the auto-encoder and GAN for $\mathcal{U}_i$ using his own dataset $\mathcal{D}_i$, which is illustrated as Figure 1(b). $\mathcal{U}_i$ first trains the auto-encoder and uses the Enc to compute the representations of $\mathcal{D}_i$, which is the input to train the GAN. Lines 7 - 9 show the process of requesting $\mathcal{D}_s$ for $\mathcal{U}_i$ from his neighbors and the process is shown as Figure 1(a). $\mathcal{U}_i$ first sends his Enc and D_1 to his neighbors. For each $\mathcal{U}_j$ in the neighbors, it computes the representations of his data by Enc and uses D_1 to obtain probability score p. $\mathcal{U}_j$ collects data with score $p > \tau$, puts into $\mathcal{D}_s$ and sends back to $\mathcal{U}_i$. Lines 13 - 14 show the adversarial training process of $\mathcal{U}_i$ using $\mathcal{D}_i$ and $\mathcal{D}_s$, which is shown as Figure 1(c). The adversarial training can reduce the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$, which can further improve the performance of f_i compared to the training of f_i directly over $\mathcal{D}_i$ and $\mathcal{D}_s$.

We explain the motivation and necessity of the adversarial training. As shown in Figure 2(a), the data in $\mathcal{D}_s$ ($\mathcal{D}_i$) are represented by circular (triangular) dots. In raw data space, $\mathcal{D}_s$ and $\mathcal{D}_i$ are partially overlapped and there exists a distribution discrepancy between them. Consequently, the model performance may not be optimal if $\mathcal{U}_i$ directly uses them to train the personalized model. However, if we can transform the raw data into another space where $\mathcal{D}_i$ and $\mathcal{D}_s$ can be well overlapped as shown in Figure 2(b), then the distribution discrepancy in this new space will be reduced and these new transformed data can be used to build a more accurate model. To minimize the distribution discrepancy between $\mathcal{D}_s$ and $\mathcal{D}_i$, we present an adversarial learning framework as shown in Figure 1(c). The adversarial learning simultaneously reduces the distribution discrepancy between $\mathcal{D}_s$ and $\mathcal{D}_i$ in the feature space and trains the model with these feature data simultaneously.

Algorithm 1 AdvPL: Adversarial Personalized Learning

Input:
$\mathcal{U}_i$ with dataset $\mathcal{D}_i$ and threshold τ;
Output:
Well trained personalized model f_i;
1: Initialize parameters in auto-encoder (Enc, Dec) and GAN(G, D_1) for $\mathcal{U}_i$;
2: Initialize parameters in feature extractor (F), model classifier (C), and discriminator (D_2) for adversarial training;
3: **/* Train auto-encoder and GAN */**:
4: $\mathcal{U}_i$ trains the auto-encoder using $\mathcal{D}_i$ and updates parameters for Enc, Dec using Eq. 1, 2 and Eq. 3;
5: $\mathcal{U}_i$ trains the GAN using representations by Enc and updates parameters for G, D_1 using Eq. 4;
6: **/* Request $\mathcal{D}_s$ for $\mathcal{U}_i$ */**:
7: $\mathcal{U}_i$ sends Enc and D_1 to his neighbor users $\mathcal{U}_j$;
8: Requested dataset $\mathcal{D}_s = \emptyset$;
9: For each neighbor user $\mathcal{U}_j$: encode $\mathcal{D}_j$ by Enc, compute probability score p using D_1, and put data in $\mathcal{D}_s$ with $p > \tau$;
10: **/* Adversarial training */**:
11: $\mathcal{U}_i$ trains C and F to optimal performance using $\mathcal{D}_i$;
12: **while not converged**:
13: Fix D_2 and F, and update C with loss Eq. 7 over $\mathcal{D}_i$ and $\mathcal{D}_s$;
14: Fix C, update D_2 and F with loss Eq. 6 and 5 over $\mathcal{D}_i$ and $\mathcal{D}_s$;
15: **end while**
16: **return** personalized model f_i

B. Train Auto-Encoder and GAN for $\mathcal{U}_i$

The first step of AdvPL is to train the auto-encoder and GAN for $\mathcal{U}_i$. Typically both Enc and Dec are deep neural networks. The Enc takes the data in $\mathcal{D}_i$ and obtains its hidden representation. The hidden representation is then fed into the Dec to output a reconstructed input. The advantage of using representations by the auto-encoder is to make the training of the GAN easier, especially for sequential data. The Enc obtains hidden representation of data in $\mathcal{D}_i$ and GAN takes it as input.

GAN has a very appealing property: its discriminator can implicitly learn hidden similarity metric and use it to discriminate real data and fake data. As a result, the discriminator is able to capture data distribution pattern of $\mathcal{U}_i$.

Encoder: The encoder is a neural network which takes mth data X_m and outputs a hidden representation as:

$$\mathbf{h}_m^{Enc} = Enc(X_m), \tag{1}$$

where $\mathbf{h}_m^{Enc}$ is the hidden representation of X_m and $Enc(\cdot)$ denotes the computation of hidden representation by the encoder neural network. The hidden representation is deemed to capture the information of the input data, which will be the input of the decoder.

Decoder: The decoder is used to reconstruct original input based on $\mathbf{h}_m^{Enc}$ and the reconstruction of X_m is expressed as:

$$X'_m = Dec(\mathbf{h}_m^{Enc}), \tag{2}$$

where X'_m is the reconstructed data of X_m and $Dec(\cdot)$ denotes the computation of reconstructed input by the decoder neural network.

The performance of the auto-encoder is evaluated by the distance between X_m and X'_m and the loss over $\mathcal{D}_i$ is expressed as:

$$\mathcal{L}_{AE} = \frac{1}{M_i}\sum_{m=1}^{M_i}(X_m - X'_m)^2 \tag{3}$$

After the training process of auto-encoder, $\mathcal{U}_i$ can use the Enc to transform input data into hidden representation and train the GAN based on the representation data. The GAN consists of a generator G and a discriminator D_1. The goal of G is to generate fake representation $\mathbf{h}_{fake}^{Enc}$ and D_1 is to distinguish real representation $\mathbf{h}_{real}^{Enc}$ and fake representation $\mathbf{h}_{fake}^{Enc}$. The objective function of the GAN is the same as the traditional GAN in previous work [7]:

$$\begin{aligned}\min_G \max_{D_1} \mathbb{E}_{\mathbf{h}_{real}^{Enc}\sim P_{data}} &\log D_1(\mathbf{h}_{real}^{Enc}) \\ &+ \mathbb{E}_{\mathbf{h}_{fake}^{Enc}\sim P(\mathbf{G})}\log(1 - D_1(G(\mathbf{h}_{fake}^{Enc}))),\end{aligned} \tag{4}$$

where P_{data} ($P_{\mathbf{G}}$) denotes the probability distribution of $\mathcal{D}_i$ (noise).

C. Adversarial Learning for Personalized Model

A straightforward approach is to directly incorporate $\mathcal{D}_s$ to his personalized model. However, the possible distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$ shown as Figure 2(a) will impede the personalized model from achieving optimal performance. To minimize the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$, we design an adversarial learning approach as shown in Figure 1(c).

The framework of adversarial training for personalized model is composed of three parts including feature representation extractor (F), discriminator D_2 and model classifier C. It should be mentioned that F and C are two parts of a complete neural network model, so the extracted representations by F can be directly used as intermediate input for C. F is used to extract representations of data in $\mathcal{D}_i$ and $\mathcal{D}_s$. The adversarial training process is analogous to the traditional GAN. The role of F is to mimic the function of the generator in GAN while the role of the discriminator D_2 is to distinguish the feature representations between $\mathcal{D}_i$ and $\mathcal{D}_s$ created by F. In our proposed AdvPL, F is trained in a manner that maps the data in $\mathcal{D}_i$ and $\mathcal{D}_s$ to feature representations with binary labels, where the label is 1 if feature representation belongs to $\mathcal{D}_i$ and is 0, otherwise. The key of the AdvPL is that F and D_2 are trained together through the adversarial learning process. More specifically, the goal of F is to generate the feature representations of data in $\mathcal{D}_i$ and $\mathcal{D}_s$ into the same space. F is to fool D_2 and makes D_2 unable to distinguish the representations between $\mathcal{D}_i$ and $\mathcal{D}_s$. On the contrary, D_2 is trained to predict whether the feature representations are from $\mathcal{D}_i$ or $\mathcal{D}_s$.

The loss function of F is similar to the generator in traditional GAN and has the following expression:

$$\mathcal{L}_F = -\mathbb{E}_{Z\sim\mathcal{D}_s}\log D_2(F(Z)) \tag{5}$$

where $F(Z)$ is the representation of data in $\mathcal{D}_s$.

The discriminator D_2 is trained to identify whether a data sample is from $\mathcal{D}_i$ or $\mathcal{D}_s$ given the feature representations. It is obvious that if the transformed representation suffers from huge distinction between $\mathcal{D}_i$ and $\mathcal{D}_s$, then D_2 is able to separate them easily. The adversarial loss of D_2 is:

$$\mathcal{L}_{D_2} = -\mathbb{E}_{X\sim\mathcal{D}_i}\log D_2(F(X)) - \mathbb{E}_{Z\sim\mathcal{D}_s}\log(1 - D_2(F(Z))) \tag{6}$$

where $F(X)$ denotes the representations of his own data in $\mathcal{D}_i$. It can be seen that D_2 plays the same role as the traditional discriminator in traditional GAN.

The combination of $\mathcal{L}_F$ and $\mathcal{L}_{D_2}$ can mimic the adversarial training process of traditional GAN. Through the adversarial learning process, the data in $\mathcal{D}_i$ and $\mathcal{D}_s$ are transformed into the hidden representation space. In this space, D_2 cannot tell the difference between them so that the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$ can be minimized.

Different from the traditional GAN which only generates realistic examples, our ultimate goal is to train a more accurate model for individual user. As shown in Figure 1(c), the extracted representations from $\mathcal{D}_i$ and $\mathcal{D}_s$ will be fed into C

and the loss function of C with K classes is:

$$\begin{aligned}\mathcal{L}_C = &- \sum_{m=1}^{M_i} \sum_{k=1}^{K} {}_{k=Y_m} \log C(F(X_m)) \\ &- \sum_{m=1}^{|\mathcal{D}_s|} \sum_{k=1}^{K} {}_{k=Y_m} \log C(F(Z_m)),\end{aligned} \tag{7}$$

where C is the final layer of model classifier and Y_m are the corresponding labels. M_i is the data number of $\mathcal{D}_i$ and $|\mathcal{D}_s|$ is the size of $\mathcal{D}_s$. Therefore, the full framework is to minimize the joint loss function $\mathcal{L}$:

$$\mathcal{L} = \mathcal{L}_C + \mathcal{L}_F + \mathcal{L}_{D_2}, \tag{8}$$

where $\mathcal{L}$ denotes the sum loss of the adversarial learning framework. The joint training process of the adversarial learning framework is summarized in Lines 13 - 14 in Algorithm 1. It mainly includes two parts. The first part is that $\mathcal{U}_i$ trains C and F using $\mathcal{D}_i$. Our goal is to build a personalized model and use similar data $\mathcal{D}_s$ to improve model performance, so we first achieve optimal model performance based on $\mathcal{D}_i$. The second step is to alternatively train C, F and D_2. To train C, we fix F and D_2. To train F and D_2, we fix C. In this way, the balance of the adversarial training will be under better control. To be noted here, we first train the personalized model to optimal performance before starting the adversarial training. Compared to training D_2 at the beginning, it is easier to improve the performance of the adversarial training.

IV. Experimental Results

In this section, we evaluate the performance of AdvPL in the following three aspects. First, we compare the performance of the personalized model and global model to demonstrate the benefits of the personalized model. Second, we measure the performance improvement of the personalized model with requested similar data and adversarial training. Third, we compare our algorithm with other similarity metrics and demonstrate the advantages of our proposed framework.

A. Experimental Setup

Datasets: To evaluate the performance of our algorithm, we conduct our experiments on two real-world datasets, including UNIX Command Sequence and Shakespeare Text. The UNIX Command Sequence dataset [30] is composed of 50 files, where each file corresponds to one user's command sequence collected by the UNIX acct auditing mechanism. Each user is recorded with a long sequence consisting of the UNIX command in a period of time, such as *troff, dpost, eqn, sed, cat, ls, gs* and so forth. The sequence length of each user is 15000 and the average number of command types for these 50 users is over 100. We split the long command sequence of each user evenly into 500 sequences. For each sequence, we aim to build a classifier that predicts the final command based on the previous commands in this sequence. More specifically, the input data is a sequence consisting of UNIX commands with length T and the task is to predict the next command at the $T+1$ step. The Shakespeare Text dataset is constructed from The Complete Works of William Shakespeare [13]. This dataset is written in the form of plays and each speaking role in the plays is treated as an individual user. We subsample 40 speaking roles and build one personalized model for each speaking role. For each speaking role, we process its input text data to a sequence list where each sequence has a fixed length 100. The numbers of sequences of each speaking role are within the range between 5000 and 10000. The task is to predict the next character after reading previous characters in each sequence.

Hyperparameters: In our experiment, we use two-layer long short-term memory (LSTM) neural networks as the classification model for both two tasks. Each layer of the LSTM classification model has the dimension 256 and the output of the second LSTM layer is sent to a softmax output layer for prediction. For each user, we select 80% of the sequences as training data and the rest as testing data. We also choose LSTM to build the auto-encoder which is composed of an encoder and a decoder. Both the encoder and a decoder consist of two-layer LSTMs. The dimensions of the hidden layer in the encoder and decoder are both set as 256. Each subsequence is embedded with 512 dimensions before sending to the encoder as input. The last hidden layer of the encoder is taken out to be the input of the decoder. For the GAN model, both the discriminator and generator are feedforward neural networks. More specifically, the generator has two hidden layers with dimensions 50 and 100, respectively. The discriminator also has two hidden layers with dimensions 100 and 50, respectively. The dimension of the Gaussian noise is the same as the size of the hidden representations by the encoder.

Baselines: In our proposed AdvPL, each user collects $\mathcal{D}_s$ from other users and trains his personalized model using $\mathcal{D}_i$ and $\mathcal{D}_s$ in an adversarial training manner. We compare our AdvPL with the following baselines:

- Global: the global model is a one-fit-all model for all of the users, which is built on the collection of all users' training data.
- PL: each user trains his own personalized model (the same structure as the global model) only based on his own training data.
- PL_Sim: each user collects $\mathcal{D}_s$ from other users using our proposed auto-encoder and GAN and trains the model with the combination of $\mathcal{D}_i$ and $\mathcal{D}_s$.
- PL_Rand: each user randomly requests $\mathcal{D}_s$ and trains the model same as PL_Sim.
- PL_Euc: each user requests $\mathcal{D}_s$ from other users based on the euclidean similarity metric and trains his model as PL_Sim. The euclidean similarity metric is computed as follows: for each user, we compute the one-hot-vector of each command and obtain a vector v_i by averaging the one-hot-vector of all commands. Similarly, for each subsequence of other users, we compute its vector v_s. Then we can compute the euclidean similarity metric between v_i and v_s.
- PL_Cos: each user requests $\mathcal{D}_s$ from other users based on the cosine similarity metric and trains his model as

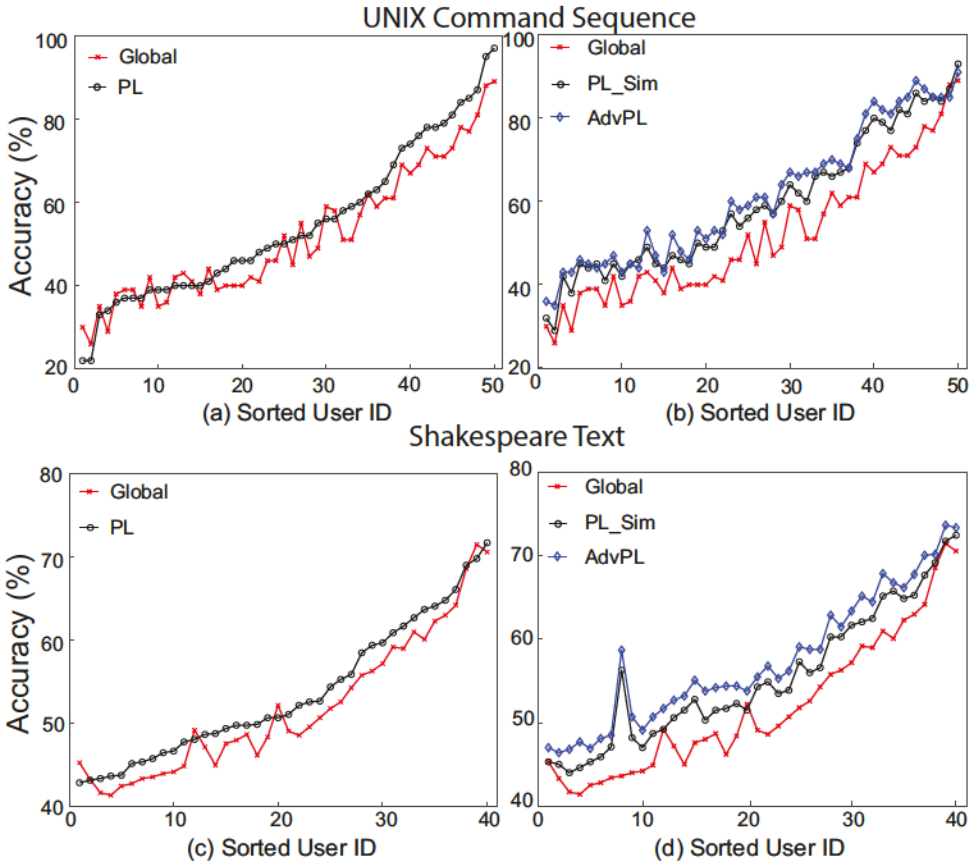


Fig. 3: UNIX Command Sequence(a): The accuracy of the Global and PL for each user. (b): The accuracy of the Global, PL_Sim and PL_Adv for each user. Shakespeare text(c): The accuracy of the Global and PL for each user. (d): The accuracy of the Global, PL_Sim and PL_Adv for each user.

PL_Sim. The computation of the cosine similarity metric is the same as PL_Euc.

- PL_Multi: each user requests $\mathcal{D}_s$ using our proposed autoencoder and GAN. We apply the multi-task framework [31] and treat $\mathcal{D}_i$ and $\mathcal{D}_s$ as two different tasks. In this implementation, only the final classification layer for $\mathcal{D}_i$ and that for $\mathcal{D}_s$ are different.

B. Performance Comparison Between Global and PL

In this experiment, we compare the performance of the Global and PL for each user to show the necessity of building the personalized model. The experimental result for UNIX Command Sequence (Shakespeare Text) is shown as Figure 3(a)(Figure 3(c)). To illustrate the result more clearly, we sort the users according to the prediction accuracy of the personalized model. The accuracy of each user using the Global (PL) is shown as the red star line (black dot line). It can be seen that the trend of the PL is above the Global for most users. More specifically, the average accuracy of the Global and PL for UNIX Command Sequence (Shakespeare Text) is 51.98% (51.87%) and 54.86% (53.67%), respectively. The maximum accuracy increment of the PL over the Global for UNIX Command Sequence (Shakespeare Text) is 8.0% (3.7%). As aforementioned, Global captures the overall information of all training samples and overlooks the user-specific information. However, the user-specific information is the key to improve the performance of the PL. Hence the PL can better learn the pattern of each user.

C. Performance Comparison of PL_Sim and AdvPL

In this experiment, we test the effectiveness of the proposed algorithm and show the accuracy of the PL_Sim and AdvPL for UNIX Command Sequence (Shakespeare Text) as Figure 3(b) (Figure 3(d)). The size of requested $\mathcal{D}_s$ for UNIX Command Sequence (Shakespeare Text) is set as 1000 (8000). The accuracy of the PL_Sim (AdvPL) is shown as the black dot line (blue triangular line). For comparison, we also plot the accuracy of the Global as the red star line.

We have the following observations. First, the accuracy of the AdvPL is higher than that of the PL_Sim. It demonstrates that adversarial learning can minimize the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$. The reduced distribution discrepancy can help the AdvPL achieve higher accuracy than the PL_Sim. Second, both the PL_Sim and AdvPL have a great advantage over the Global. It shows that our proposed algorithm is effective to select similar data for each user and help improve the performance of the personalized model. More specifically, the average accuracy of the PL_Sim and AdvPL for UNIX

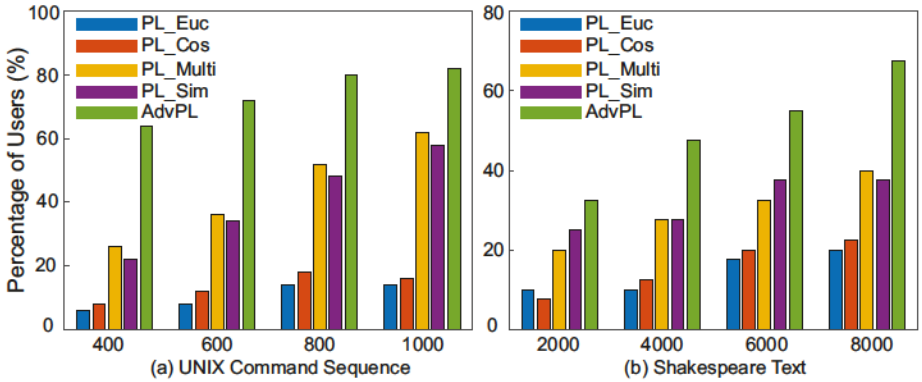


Fig. 4: The percentage of users with accuracy increment over 5% using the PL_Euc, PL_Cos, PL_Sim and AdvPL compared to the PL.

Command Sequence (Shakespeare Text) is 6.56% (3.49%) and 9.44% (5.57%) higher than that of the Global, respectively. In contrast, the average accuracy of the PL for UNIX Command Sequence (Shakespeare Text) is only 2.88% (1.49%) higher than that of the Global.

D. Performance Comparison Using Different Similarity Metrics

Previous section only shows the performance between the PL_Sim and AdvPL. In this section, we compare the performance of the personalized model using other similarity metrics. For better comparison, we show the average accuracy rather than plotting the accuracy of all users. The comparison result for UNIX Command Sequence (Shakespeare Text) is shown as the 5th row in Table I (Table II) with $|\mathcal{D}_s| = 1000$ ($|\mathcal{D}_s| = 8000$). We have the following interesting observations. First, if the user randomly requests $\mathcal{D}_s$, the average performance of the PL_Rand is slightly decreased compared to the PL. Second, the performance of the PL_Euc and PL_Cos helps improve the performance over the PL, however, the performance increment of our proposed PL_Sim is the best among these three similarity metrics. Third, the AdvPL achieves the best performance among all strategies. PL_Multi adopts the general multi-task learning framework [31] to learn common features between $\mathcal{D}_i$ and $\mathcal{D}_s$, however, it only achieves similar performance as PL_Sim, which demonstrates that our proposed AdvPL is more effective than the traditional multi-task learning method.

Moreover, we also study the effect of requested dataset $\mathcal{D}_s$ size on the performance of the PL_Sim and AdvPL. The result for UNIX Command Sequence (Shakespeare Text) is shown in Table I (Table II) with $\mathcal{D}_s$ size increasing from 400 to 1000 (2000 to 8000). We have the following three observations from the results. First, the performance of the PL_Rand is slightly decreased compared to the PL under different $\mathcal{D}_s$ sizes. It is reasonable that large distribution discrepancy of the randomly requested $\mathcal{D}_s$ and $\mathcal{D}_i$ can deteriorate the personalized model performance. Second, the accuracy values of the PL_Euc, PL_Cos, PL_Multi and PL_Sim increase with the increasing size of $\mathcal{D}_s$. Third, we discover that with smaller $\mathcal{D}_s$ size, the accuracy increment from PL_Sim to AdvPL is more significant. More specifically, the accuracy increment of the AdvPL over the PL_Sim for UNIX Command Sequence (Shakespeare Text) is 3.36% (2.39%), 2.90% (2.14%), 2.56% (1.62%) and 2.20% (1.38%), respectively, with the corresponding $\mathcal{D}_s$ size as 400 (2000), 600 (4000), 800 (6000), 1000 (8000). The reason is that with smaller $\mathcal{D}_s$ size, some less similar data in $\mathcal{D}_s$ impedes the model performance improvement to a greater extent due to the smaller total amount of data. In contrast, if the total data size is larger, then the effects of some less similar data is smaller and the advantage of the AdvPL is also weakened. As a result, the AdvPL plays a more important role for personalized model with smaller budget size of $\mathcal{D}_s$. It is more practical in real scenarios that one user can only request limited amount of data from other users due to the privacy concern or communication cost burden.

To further compare the performance of different strategies, we plot the percentage of users with accuracy increment greater than 5% using the PL_Euc, PL_Cos, PL_Multi, PL_Sim and AdvPL over the PL under different $\mathcal{D}_s$ size. The result is shown in Figure 4. It can be seen that the AdvPL greatly outperforms other strategies. In addition, the PL_Sim achieves better performance than the PL_Euc and PL_Cos, demonstrating the effectiveness of our proposed method in requesting similar data.

E. Effects of Probability Distribution and Budget Size for PL_Sim and AdvPL

In this section, we investigate the effects of the probability distribution of the requested data determined by D_2 and the budget size of $\mathcal{D}_s$ for the PL_Sim and AdvPL. We randomly select a single user (ID = 29) in the UNIX Command Sequence dataset and conduct the experiments for demonstration purpose. Figure 5(a) shows the effects of probability distribution of the requested data for the PL_Sim and AdvPL. We divide the range of probability score by D_2 evenly into five regions between 0.5 and 1.0. For each region, we select $\mathcal{D}_s$ containing 1000 samples and test the accuracy of the PL_Sim and AdvPL. For example, if we select the range [0.5, 0.6], it means the probability score of all requested data falls into the range [0.5, 0.6]. The accuracy of the PL_Sim (AdvPL) is shown as the black dot line (red star line).

TABLE I: Performance comparison for UNIX Command Sequence using different similarity metrics and $\mathcal{D}_s$ sizes. The average accuracy of the Global and PL is: 51.98% 54.86%, respectively. The scale of the numbers in Table I is %.

$\lvert\mathcal{D}_s\rvert$	PL_Rand	PL_Euc	PL_Cos	PL_Multi	PL_Sim	AdvPL
400	54.48	55.12	55.34	56.84	56.78	**60.14**
600	54.24	55.28	55.62	57.96	57.78	**60.68**
800	54.62	55.56	55.78	58.62	58.54	**61.10**
1000	54.46	55.88	56.10	59.46	59.22	**61.42**

TABLE II: Performance comparison for Shakespeare Text using different similarity metrics and $\mathcal{D}_s$ sizes. The average accuracy of the Global and PL is: 51.88% 53.35%, respectively. The scale of the numbers in Table II is %.

$\lvert\mathcal{D}_s\rvert$	PL_Rand	PL_Euc	PL_Cos	PL_Multi	PL_Sim	AdvPL
2000	52.36	53.63	53.45	54.21	54.23	**56.62**
4000	52.49	53.65	53.61	54.70	54.77	**56.91**
6000	52.35	53.79	53.80	55.24	55.69	**57.31**
8000	52.60	53.91	53.88	55.49	56.07	**57.45**

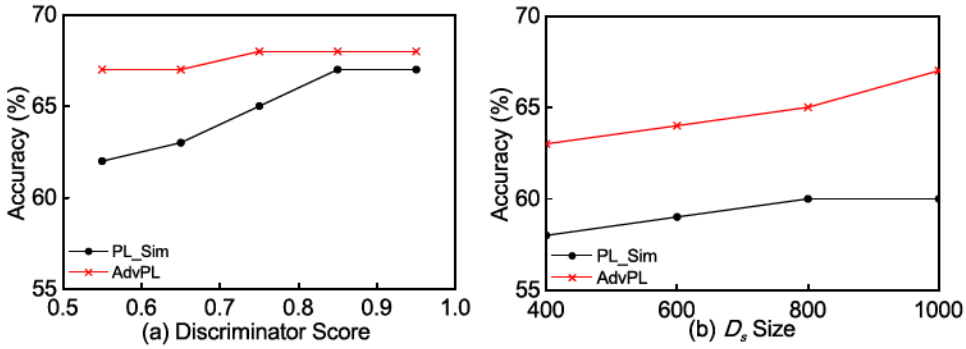


Fig. 5: (a): The accuracy of the PL_Sim and AdvPL of a single user (UNIX Command Sequence) based on $\mathcal{D}_s$ with different probability distribution of D_2. (b): The accuracy of the PL_Sim and AdvPL based on $\mathcal{D}_s$ with different size. The PL accuracy of the single user (ID = 29) is 57%.

Three noteworthy points can be drawn from the result. First, the accuracy of the PL_Sim is lower if the probability distribution of the data in $\mathcal{D}_s$ is in a low range. For example, if all of the data in $\mathcal{D}_s$ is requested within the range [0.9, 1.0], then the accuracy of the PL_Sim is 5% higher than that of the $\mathcal{D}_s$ within the range [0.5, 0.6]. As we know, higher probability by the discriminator indicates the data is more likely to be sampled from $\mathcal{D}_i$. Consequently, lower probability distribution range of $\mathcal{D}_s$ will cause larger distribution discrepancy with $\mathcal{D}_s$ and lower performance improvement of the PL_Sim. Second, the accuracy of the AdvPL for $\mathcal{D}_s$ with different probability distribution ranges is stable. The advantage of the AdvPL is that it can reduce the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$, so that the performance of the AdvPL can still be improved greatly under a higher distribution discrepancy. It demonstrates that the AdvPL can better improve the personalized model performance if only less similar data is available. Third, the advantage of the AdvPL is weakened if the distribution discrepancy between $\mathcal{D}_i$ and $\mathcal{D}_s$ is smaller. The reason is that if $\mathcal{D}_s$ is of high similarity compared with $\mathcal{D}_i$, then the user can directly combine them and train the model without considering the effect of distribution discrepancy.

Figure 5(b) shows the effects of $\mathcal{D}_s$ size on the performance of the PL_Sim and AdvPL. To test the sensitivity of the adversarial training process, we request less similar data and set the probability distribution of the data in $\mathcal{D}_s$ within the range [0.5, 0.6]. The black dot line (red star line) shows the performance of the PL_Sim (AdvPL). First, the accuracy of the PL_Sim increases slowly with the increasing size of $\mathcal{D}_s$ as we select less similar data with the probability distribution in the low range [0.5, 0.6]. So although the user requests more data, the performance gain by these increasing less similar data is still insignificant. Second, with the adversarial training, the accuracy of the AdvPL improves greatly over the PL_Sim. It can be seen that the performance improvement of the AdvPL is almost stable under different sizes of $\mathcal{D}_s$. It demonstrates that the AdvPL can still improve the model performance greatly even with a limited budget size and less similar data available.

V. Conclusions and Future Work

In this paper, we proposed the AdvPL framework enabling individual users to effectively collect data from other users and train a robust personalized model. We proposed to use the auto-encoder and GAN to select similar data from other users. The trained auto-encoder and GAN, which capture inherent information of user's personal data, can be efficiently used to choose similar data from others, thereby avoiding

tedious process of paired data comparison. We also developed a novel adversarial training that maps selected data and user's own data to the same feature space and jointly train the personalized model. The adversarial training can effectively mitigate potential distribution discrepancy between selected data and user's own data. We conducted extensive experiments to demonstrate the effectiveness of the proposed framework.

There are two major directions for our future work. First, we will study how to achieve privacy in our AdvPL. The auto-encoder and GAN contain private information of the individual user's data. Previous works demonstrate that deep learning models can memorize abundant information of the training data [32]. To protect the privacy of training data in $\mathcal{D}_i$, we can build differential privacy preserving versions of auto-encoder and GAN, e.g., by adopting the ideas of [33] and [34] respectively. Moreover, the data $\mathcal{D}_s$ collected from other users are also private and users may not want to share. We will study the use of local differential privacy [35] for private data comparison and collection. Second, we will investigate how to determine most appropriate data to improve the performance of personalized model. Our current work is based on the idea of selecting similar data to boost the performance. Ideally, we want to determine the properties of new data that can best improve the model performance, e.g., reducing the prediction error of the built model. With that, we only need to collect truly useful data and skip redundant ones, which will greatly reduce the communication burden and improve efficiency. One idea is to use active learning to select new data that improve personalized model performance. However, existing active learning strategies [36] may not be directly applied here because the data determined by the active learning may contain large distribution discrepancy from individual user's own data. It is interesting to investigate how to combine the active learning and personalized model.

VI. Acknowledgments

This work was supported in part by NSF 1920920, 1937010, 1940093, and 1946391.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.

[2] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *AAAI*, 2015.

[3] X. Dong, L. Yu, Z. Wu, Y. Sun, L. Yuan, and F. Zhang, "A hybrid collaborative filtering model with deep structure for recommender systems," in *AAAI*, 2017.

[4] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *NeurIPS*, 2012.

[5] D. H. Park, H. K. Kim, I. Y. Choi, and J. K. Kim, "A literature review and classification of recommender systems research," *Expert systems with applications*, 2012.

[6] Y. Cheng, F. Wang, P. Zhang, and J. Hu, "Risk prediction with electronic health records: A deep learning approach," in *SDM*, 2016.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS*, 2014.

[8] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *OSDI*, 2014.

[9] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *NeurIPS*, 2017.

[10] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan, "Adacomp: Adaptive residual gradient compression for data-parallel distributed training," in *AAAI*, 2018.

[11] S. Wang, A. Pi, X. Zhao, and X. Zhou, "Scalable distributed dl training: Batching communication and computation," in *AAAI*, 2019.

[12] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *NeurIPS*, 2018.

[13] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2016.

[14] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *ACM CCS*, 2017.

[15] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *NeurIPS*, 2017.

[16] C. Che, C. Xiao, J. Liang, B. Jin, J. Zho, and F. Wang, "An rnn architecture with dynamic temporal matching for personalized predictions of parkinson's disease," in *SDM*, 2017.

[17] Q. Suo, F. Ma, Y. Yuan, M. Huai, W. Zhong, A. Zhang, and J. Gao, "Personalized disease prediction using a cnn-based similarity learning method," in *IEEE BIBM*, 2017.

[18] E. Choi, M. T. Bahadori, E. Searles, C. Coffey, M. Thompson, J. Bost, J. Tejedor-Sojo, and J. Sun, "Multi-layer representation learning for medical concepts," in *ACM KDD*, 2016.

[19] M. Huai, C. Miao, Q. Suo, Y. Li, J. Gao, and A. Zhang, "Uncorrelated patient similarity learning," in *SDM*, 2018.

[20] F. Wang, J. Sun, and S. Ebadollahi, "Composite distance metric integration by leveraging multiple experts' inputs and its application in patient similarity assessment," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2012.

[21] F. Luo, G. Ranzi, X. Wang, and Z. Y. Dong, "Social information filtering-based electricity retail plan recommender system for smart grid end users," *IEEE Transactions on Smart Grid*, 2017.

[22] P. Kouki, S. Fakhraei, J. Foulds, M. Eirinaki, and L. Getoor, "Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems," in *ACM RecSys*, 2015.

[23] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE TPAMI*, 2013.

[24] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," in *IEEE CVPR*, 2015.

[25] A. H. Liu, Y.-C. Liu, Y.-Y. Yeh, and Y.-C. F. Wang, "A unified feature disentangler for multi-domain image translation and manipulation," in *NeurIPS*, 2018.

[26] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," in *ICLR*, 2017.

[27] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *IEEE CVPR*, 2016.

[28] D. Bouchacourt, R. Tomioka, and S. Nowozin, "Multi-level variational autoencoder: Learning disentangled representations from grouped observations," in *AAAI*, 2018.

[29] S. Narayanaswamy, T. B. Paige, J.-W. Van de Meent, A. Desmaison, N. Goodman, P. Kohli, F. Wood, and P. Torr, "Learning disentangled representations with semi-supervised deep generative models," in *NeurIPS*, 2017.

[30] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theusan, Y. Vardi *et al.*, "Computer intrusion: Detecting masquerades," *Statistical science*, 2001.

[31] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[32] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *ACM CCS*, 2017.

[33] N. Phan, Y. Wang, X. Wu, and D. Dou, "Differential privacy preservation for deep auto-encoders: an application of human behavior prediction," in *AAAI*, 2016.

[34] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou, "Differentially private generative adversarial network," *CoRR*, 2018.

[35] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy and statistical minimax rates," in *IEEE FOCS*, 2013.

[36] B. Settles, "Active learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.