Continuous Learning for Large-scale Personalized Domain Classification

Han Li[1], **Jihwan Lee**[2], **Sidharth Mudgal**[2], **Ruhi Sarikaya**[2], and **Young-Bum Kim**[2]

[1]University of Wisconsin, Madison
hanli@cs.wisc.edu
[2]Amazon Alexa AI
{jihwl, sidmsk, rsarikay, youngbum}@amazon.edu

Abstract

Domain classification is the task of mapping spoken language utterances to one of the natural language understanding domains in intelligent personal digital assistants (IPDAs). This is a major component in mainstream IPDAs in industry. Apart from official domains, thousands of third-party domains are also created by external developers to enhance the capability of IPDAs. As more domains are developed rapidly, the question of how to continuously accommodate the new domains still remains challenging. Moreover, existing continual learning approaches do not address the problem of incorporating personalized information dynamically for better domain classification. In this paper, we propose CoNDA, a neural network based approach for domain classification that supports incremental learning of new classes. Empirical evaluation shows that CoNDA achieves high accuracy and outperforms baselines by a large margin on both incrementally added new domains and existing domains.

1 Introduction

Domain classification is the task of mapping spoken language utterances to one of the natural language understanding (NLU) domains in intelligent personal digital assistants (IPDAs), such as Amazon Alexa, Google Assistant, and Microsoft Cortana, etc. (Sarikaya, 2017). Here a domain is defined in terms of a specific application or functionality such as weather, calendar or music, which narrows down the scope of NLU. For example, given an utterance "*Ask Uber to get me a ride*" from a user, the appropriate domain would be one that invokes the "Uber" app.

Traditionally IPDAs have only supported dozens of well-separated domains, where each is defined in terms of a specific application or functionality such as calendar and weather (Sarikaya et al., 2016; Tur and De Mori, 2011; El-Kahky et al., 2014). In order to increase the domain coverage and extend the capabilities of the IPDAs, mainstream IPDAs released tools to allow third-party developers to build new domains. Amazons Alexa Skills Kit, Googles Actions and Microsofts Cortana Skills Kit are examples of such tools. To handle the influx of new domains, large-scale domain classification methods like SHORTLISTER (Kim et al., 2018b) have been proposed and have achieved good performance.

As more new domains are developed rapidly, one of the major challenges in large-scale domain classification is how to quickly accommodate the new domains without losing the learned prediction power on the known ones. A straightforward solution is to simply retraining the whole model whenever new domains are available. However, this is not desirable since retraining is often time consuming. Another approach is to utilize continual learning where we dynamically evolve the model whenever a new domain is available. There is extensive work on the topic of continual learning, however there is very little on incrementally adding new domains to a domain classification system.

To mitigate this gap, in this paper we propose the CoNDA solution for continuous domain adaptation. Given a new domain, we keep all learned parameters, but only add and update new parameters for the new domain. This enables much faster model updates and faster deployment of new features to customers. To preserve the learned knowledge on existing domains to avoid the notorious catastrophic forgetting problem (Kemker et al., 2018), we propose cosine normalization for output prediction and domain embedding regularization for regularizing the new domain embedding. Also, we summarize the data for existing domains by sampling exemplars, which will be used together

with the new domain data for continuous domain adaptation. This is shown to further alleviate the overfitting on the new domain data. Empirical evaluation on real data with 900 domains for initial training and 100 for continuous adaptation shows that CoNDA out performs the baselines by a large margin, achieving 95.6% prediction accuracy on average for the 100 new domains and 88.2% accuracy for all seen domains after 100 new domains have been accommodated (only 3.6% lower than the upperbound by retraining the model using all domain data). To summarize, we make the following contributions in this paper:

- We introduce the problem of continuous domain adaptation for large-scale personalized domain classification.
- We describe CoNDA, a new solution for continuous domain adaptation with Cosine normalization, domain embedding regularization and negative exemplar sampling techniques. Our solution advances the research in continuous domain adaptation.
- We conduct extensive experiments showing that CoNDA achieves good accuracy on both new and existing domains, and outperforms the baselines by a large margin.

2 Background and Problem Definition

2.1 Domain Classification

Domain classification is the task of mapping spoken language utterances to one of the NLU domains in IPDAs. A straightforward solution to tackle this problem is to ask users to explicitly mention the domain name with a specific invocation pattern. For example, for the utterance "*Ask Uber to get me a ride*", the invocation pattern is "*Ask* {*domain*} *to* {*perform action*}". While it makes things much simpler for the domain classifier, this significantly limits natural interaction with IPDAs as users need to remember the domain names as well as the invocation pattern. To address this limitation, *name-free domain classification* methods were developed for more user friendly interactions, and have been getting more attention recently. We specifically focus on the name-free scenario in this paper.

2.2 The Shortlister System

To our knowledge, the state-of-the-art for name-free domain classification is SHORTLISTER (Kim et al., 2018b), which leverages personalized user provided information for better classification performance. Specifically, it contains three main modules.

The first module is the LSTM-based encoder to map an utterance to a dimension-fixed vector representation. Given an utterance, each word is first represented as dense vectors using word embeddings, then a bidirectional LSTM (Graves and Schmidhuber, 2005) is be used to encode the full utterance.

The second module is the personalized domain summarization module. For each utterance from an IPDA user, a list of domains have been enabled by the user. These enabled domains can be viewed as user-specific personalized information. It has been shown that the domain classification accuracy can be significantly improved by leveraging information about enabled domains (Kim et al., 2018b). To represent the domain enablement information, first each enabled domain is mapped to a fixed-dimensional embedding, then a summarization vector is generated by taking an attention weighted sum (Luong et al., 2015) over the enabled domain embeddings.

Once the utterance representation and the enabled domain summarization are calculated, we concatenate the two vectors as the final representation. Then the third module, a feed-forward network, is used to predict the confidence score with a sigmoid function for each domain.

2.3 Continuous Domain Adaptation

As more new domains are developed, a major challenges in large-scale domain classification is quickly accommodating the new domains into the live production domain classification model without having to perform a full retrain. We refer to this problem as *Continuous Domain Adaptation* (CDA). In this paper, we specifically focus on the case of purely online learning where new domains where added one by one, since in practice we want to quickly integrate a new domain into the system as soon as it becomes available. We formally define the problem below.

Definition 1 *(Online continuous domain adaptation) Given a collection of k domains $S_k = \{s_1, s_2, \ldots, s_k\}$, suppose we have a dataset $\mathcal{D}_k$ defined on S_k where each item is a triple (u, s, E) with the utterance $u \in U$ (the set for all possible utterances), the ground-truth domain $s \in S_k$, and*

the enabled domains $E \subseteq S_k$. Denote $\mathcal{P}(S_k)$ as the powerset of S_k, a model $M_k : U \times \mathcal{P}(S_k) \rightarrow S_k$ has been trained on $\mathcal{D}_k$ for domain classification with the accuracy $M_k(\mathcal{D}_k)$. At some point, a new domain s_{k+1} is available with the corresponding dataset $D_{k+1} = \{(u, s_{k+1}, E) \mid E \subseteq S_{k+1}\}$ with $S_{k+1} = S_k \cup \{s_{k+1}\}$. Taking advantage of D_{k+1}, the continuous adaptation for s_{k+1} is to update M_k to $M_{k+1} : U \times \mathcal{P}(S_{k+1}) \rightarrow S_{k+1}$ so that the model can make predictions for s_{k+1}, with the goal of maximizing $M_{k+1}(D_{k+1})$ and minimizing $M_k(\mathcal{D}_k) - M_{k+1}(\mathcal{D}_k)$.

3 The CoNDA Solution

We introduce CoNDA (**Co**ntinuous **N**eural **D**omain **A**daptation), a variation of SHORTLISTER that is capable of handling online CDA decribed in Definition 1. Similar to SHORTLISTER, it has three main modules.

The first module is the LSTM-based utterance encoder which shares the same architecture as the one used in SHORTLISTER, that maps an input utterance into a dense vector. After the training on the initial k-domain data $\mathcal{D}_k$, we freeze all parameters (i.e., the word embedding lookup and the bi-LSTM parameters) of this module from changing for the subsequent online domain adaptation tasks. Usually the value of k is large enough (hundreds or even thousands in real-world, at least 100 in our experiments), thus it is safe to assume that the parameters have been tuned sufficiently well to encode utterances from all existing and future domains. In this work we treat new words in the new domains as unknown and leave the problem of vocabulary expansion as future work.

The second module is the personalized domain summarization module which will map the enabled domains of an input utterance to a dense vector representation. It is also similar to the one in SHORTLISTER, except we will evolve the module as we are adding new domains. Specifically, given dataset $\mathcal{D}_k$ on k domains for initial training, a domain embedding table $T_k \in \mathbb{R}^{k \times d_s}$ will be learned where d_s is the size of the domain embeddings. When a new domain s_{k+1} is available, we expand T_k to $T_{k+1} \in \mathbb{R}^{(k+1) \times d_s}$ by: (1) freezing the learned embeddings for all known domains; (2) adding a new row $t_{k+1} \in \mathbb{R}^{d_s}$ to T_k as the domain embedding for s_{k+1} and updating the new parameters t_{k+1} using all available training data at hand (i.e., the dataset D_{k+1} and the negative samples which will be discussed later in this section). We repeat this procedure whenever a new domain is available. To avoid over-fitting on t_{k+1}, we introduce a new regularization term into the loss function. We describe the details in Section 3.2.

The third module is a two-layer feed-forward network as the classifier. The first layer $f^{(1)} : \mathbb{R}^{d_u + d_s} \rightarrow \mathbb{R}^{d_h}$ maps the concatenation of the utterance embedding (in size d_u) and domain summarization (in size d_s) into fix-sized hidden representation (in size d_h) using a fully connected layer followed by SELU activation (Klambauer et al., 2017), which is identical to the one in SHORTLISTER. Then the prediction layer $f^{(2)} : \mathbb{R}^{d_h} \rightarrow \mathbb{R}^k$ maps the hidden representation to the final domain prediction scores. Unlike SHORTLISTER where the final prediction score is the dot product of the weight vector and the hidden representation, we choose to use the cosine score of the two, referred to as *cosine normalization*. To support online CDA when a new domain is available, we apply a similar approach to the domain embedding expansion described above to expand the prediction layer. Specifically, denote $W_k^{(2)} \in \mathbb{R}^{k \times d_h}$ be the weights for the prediction layer that has been trained on the initial k domains. To adapt the new domain d_{k+1}, we expand $W_k^{(2)}$ to $W_{k+1}^{(2)} \in \mathbb{R}^{(k+1) \times d_h}$ by first freezing all learned parameters and adding a new row of learnable parameters $w_{k+1} \in \mathbb{R}^{d_h}$ to $W_k^{(2)}$.

As each time we only add one new domain, all training utterances during the update will have the same label. Thus, it's easy to overfit the new data such that catastrophic forgetting occurs. Inspired by (Rebuffi et al., 2017), we also propose a negative sampling procedure to leverage (limited) information on the known domains to alleviate the catastrophic forgetting problem. For the rest of the section, we will first talk about cosine normalization, and then domain embedding regularization, and finally negative sampling.

3.1 Cosine Normalization

As mentioned above, we use the cosine similarity of the weights and the hidden representation vector instead of the linear dot product in the prediction layer. Formally, let $f_k^{(2)} : \mathbb{R}^{d_h} \rightarrow [-1, 1]^k$ be the prediction layer for k domains with parameters $W_k^{(2)} \in \mathbb{R}^{k \times d_h}$. Given an input hidden representation $h \in \mathbb{R}^{d_h}$ from $f^{(1)}$, the score for the i-th

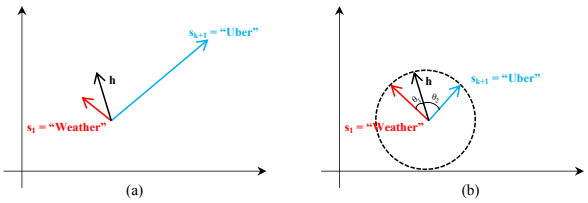

Figure 1: Cosine Example.

domain under cosine normalization is:

$$f_{k,i}^{(2)}(h) = \cos\left(h, W_{k,i}^{(2)}\right) = \frac{h \cdot W_{k,i}^{(2)}}{\|h\| \left\|W_{k,i}^{(2)}\right\|} \quad (1)$$

To understand why cosine is better in the case of online CDA, let's first see the problem with the dot-product method. Suppose we are accommodating s_{k+1} with dataset D_{k+1}, because we train the new parameters w_{k+1} only on D_{k+1} where all utterances have the same domain s_{k+1}, the model can easily get good training performance on $M_{k+1}(D_{k+1})$ by simply maximizing the values in w_{k+1} such that the dot product of the hidden representation with w_{k+1} is larger than the dot product with any other $w_i, 1 \leq i \leq k$. Effectively this leads to the model predicting domain s_{k+1} for any given utterance. Using cosine normalization instead as described in Eq. 1 removes the incentive to maximize the vector length of w_{k+1}.

Example 1 *Suppose M_k has been initially trained on $\mathcal{D}_k$, and domain s_1="Weather". Given an utterance u = "What's the weather today?", M_k correctly classifies u into s_1. Now a new domain s_{k+1}="Uber" is coming and we evolve M_k to M_{k+1}. As the norm of the weights w_{k+1} could be much larger than w_1 in the prediction layer, even if the hidden representation h of u is closer to s_1 in direction, M_{k+1} will classifier u into s_{k+1} as it has a higher score, shown in Figure 1.a. However if we measure the cosine similarity, M_{k+1} will classify u correctly because we now care more about the directions of the vectors, and the angle θ_1 between h and s_1 is smaller (representing higher similarity) than the angle θ_2 between h and s_{k+1}, as shown in Figure 1.b.*

As we use the cosine normalization, all prediction scores are mapped into the range [-1, 1]. Therefore it's not proper to use log-Sigmoid loss function as in SHORTLISTER. So accompanying with the cosine normalization, the following hinge loss function has been used instead:

$$\begin{aligned}\mathcal{L}_{hinge} = &\sum_{i=1}^{n} y_i \max\{\Delta_{pos} - o_i, 0\} \\ &+ \sum_{i=1}^{n} (1 - y_i) \max\{o_i - \Delta_{neg}, 0\}\end{aligned} \quad (2)$$

where n is the number of all domains, o_i is the predicted score for each domain, y is a n-dimensional one-hot vector with 1 in the ground-truth label and 0 otherwise. Δ_{pos} and Δ_{neg} are the hinge thresholds for the true and false label predictions respectively. The reason we use hinge loss here is that it can be viewed as another way to alleviate the overfitting on new data, as the restrictions are less by only requiring the prediction for the ground-truth to be above Δ_{pos} and false domain predictions below Δ_{neg}. Our experiments show that this helps the model get better performance on the seen domains.

3.2 Domain Embedding Regularization

In this section, we introduce the regularizations on the domain embeddings used in the personalized domain summarization module. Recall that given an utterance u with h^u as the hidden representation from the encoder and its enabled domains E, personalized domain summarization module first compares u with each $s_i \in E$ (by calculating the dot product of h^u and the domain embedding t_i of s_i) to get a score a_i, then gets the weight $c_i = \exp(a_i)/\sum_{a_j} \exp(a_j)$ for domain s_i, and finally computes the personalized domain summary as $\sum_{e_i \in E} c_i \cdot t_i$. We observed that after training on the initial dataset D_k, the domain embedding vectors tend to roughly cluster around a certain (random) direction in the vector space. Thus, when we add a new domain embedding s_{k+1} to this personalization module, the model tends to learn to move this vector to a different part of the vector space such that its easier to distinguish the new domain from all other domains. Moreover, it also increases the ℓ_2 norm of the new domain embedding t_{k+1} to win over all other domains.

Example 2 *Suppose a similar scenario to Example 1 where we have s_1 = "Weather" in S_k and a new domain s_{k+1} = "Uber". As most utterances in D_{k+1} have s_{k+1} as an enabled domain, it's easy for the model to learn to enlarge the norm of the new domain embedding t_{k+1} as well as make it close to the context of ride sharing, so that t_{k+1}*

can dominate the domain summarization. Then coordinating with the new weights w_{k+1} in the prediction layer $f^{(2)}_{k+1}$, the network can easily predict high scores s_{k+1} and fit the dataset D_{k+1}. However, when we have utterances belonging to s_1 with s_{k+1} as an enabled domain, s_{k+1} may still dominate the summarization which makes the prediction layer tends to cast those utterances to s_{k+1}. We don't observe this on the initial training on $\mathcal{D}_k$ because s_{k+1} was not visible at that time, thus cannot be used as an enabled domain. And it's even worse if s_1 is similar to s_{k+1} in concept. For example if s_1 = "Lyft", in this case the utterances of the two domains are also similar, making the dot product of t_{k+1} and the hidden representations of the s_1's utterances even larger.

To alleviate this problem, we add a new domain embedding regularization term in the loss function to constrain the new domain embedding vector length and force it to direct to a similar area where the known domains are heading towards, so that the new domain will not dominate the domain summarization. Specifically,

$$\mathcal{L}_{der} = \sum_{i=1}^{k} \lambda_i \max\{\Delta_{der} - \cos(t_{k+1}, t_i), 0\} + \frac{\lambda_{norm}}{2} \|t_{k+1}\|^2 \quad (3)$$

We call the first part of Eq. 3 on the right hand side as the *domain similarity loss* where we ask the new domain embedding t_{k+1} to be similar to known domain t_i's controlled by a Cosine-based hinge loss. As we may not need t_{k+1} to be similar to all seen domains, a coefficient λ_i is used to weight the importance each similarity loss term. In this paper we encourage t_{k+1} to be more similar to the ones sharing similar concepts (e.g. "Uber" and "Lyft"). We assume all training data are available to us, and measure the similarity of two domains by comparing their average of utterance hidden representations.

Specifically, denote $\varphi : U \rightarrow \mathbb{R}^{d_u}$ as the LSTM-encoder that will map an utterance to its hidden representation with dimension d_u. For each domain $s_i \in S_{k+1}$, we first calculate the average utterance representation on D_i

$$\widetilde{h}_i = \sum_{(u,s_i,e)\in D_i} \frac{\varphi(u)}{|D_i|} \quad (4)$$

Then we set $\lambda_i = \lambda_{dsl} \max\{\cos(\widetilde{h}_i, \widetilde{h}_{k+1}), 0\}$ with λ_{dsl} as a scaling factor.

Combining Eq. 2 and 3, the final loss function for optimization is: $\mathcal{L}_{total} = \mathcal{L}_{hinge} + \mathcal{L}_{der}$

3.3 Sampling Negative Exemplars

So far we developed our method by training only on the new data D_{k+1}, and use regularizations to prevent overfitting. However, in many real applications all of the training data, not only D_{k+1}, is actually available, but it's not affordable to retrain the full model using all data. Inspired by (Rebuffi et al., 2017), we can select a set of exemplars from the previously trained data to further improve continual adaptation.

Suppose we are handling the new domain s_{k+1} with D_{k+1}, and all data trained previously is $\mathcal{D}_k$ on k domains S_k. For each known $s_i \in S_k$, we pick N utterances from D_i as the exemplars for s_i. Denote P_i be the exemplar set for s_i and $P = \bigcup_{i=1}^{k} P_i$ be the total exemplar set. To generate each P_i, we pick the top-N utterances that are closest to the average of the utterance hidden representation. Specifically, following Eq. 4, we first get the average representation $\widetilde{h}_i$, then P_i is defined as follow:

$$P_i = \text{argmax}_{P_i \subseteq D_i, |P_i|=N} \sum_{(u,s_i,e)\in P_i} \cos\left(\varphi(u), \widetilde{h}_i\right) \quad (5)$$

If multiple candidates satisfying Eq. 5 for P_i, we randomly pick one as P_i to break the tie. Once the domain adaptation for s_{k+1} is done, we similarly generate P_{k+1} and merge it to P. We repeat this procedure for negative sampling whenever a new domain is coming later.

As we add more new domains, the exemplar set P also grows. For some new domain D_{k+1}, we may have $|P| \gg |D_{k+1}|$. In this case, the prediction accuracy on the new domain data could be very low as the model will tend to not making mistakes on P rather than fitting D_{k+1}. To alleviate this problem, when $|P| > |D_{k+1}|$, we select a subset $P' \subseteq P$ with $|P'| = |D_{k+1}|$, and P' will be used as the final exemplar set to train together with D_{k+1}. To generate P', we just randomly sample a subset from P, since it was observed to be effective in our experiments.

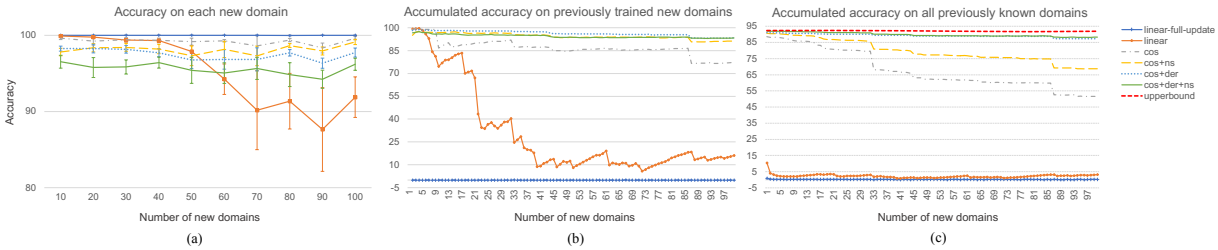

Figure 2: Overall evaluation. (a) shows the accuracy for new domains. (b) shows the accumulated accuracy for previous new domains that have been adapted to the model so far. (c) shows the accumulated accuracy for all known domains including the ones used for initial training and all previously adapted new domains.

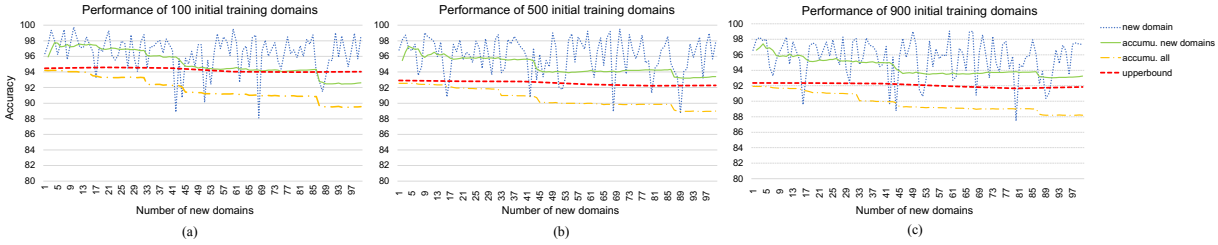

Figure 3: The model performance on different number of initial training domains. The red dashed line shows the upperbound of the accumulated accuracy, which is generated by retraining the model on all domains seen so far.

4 Empirical Evaluation

4.1 Experiment Setup

Dataset: We use a dataset defined on 1000 domains for our experiments which has 2.53M utterances, and we split them into two parts. The first part contains 900 domains where we use it for the initial training of the model. It has 2.06M utterances, and we split into training, development and test sets with ratio of 8:1:1. We refer to this dataset as "InitTrain". The second part consists of 100 domains and is used for the online domain adaptation. It has 478K utterances and we split into training, development and test sets with the same 8:1:1 ratio. We refer to this dataset as "IncTrain".

Training Setup: We implement the model in PyTorch (Paszke et al., 2017). All of the experiments are conducted on an Amazon AWS p3.16xlarge[1] cluster with 8 Tesla V100 GPUs. For initial training, we train the model for 20 epochs with learning rate 0.001, batch size 512. For the continuous domain adaptation, we add the new domains in a random order. Each domain data will be trained independently one-by-one for 10 epochs, with learning rate 0.01 and batch size 128. For both training procedures, we use Adam as the optimizer. The development data is used to pick the best model in different epoch runs. We evaluate the classification accuracy on the test set.

[1] https://aws.amazon.com/ec2/instance-types/p3/

4.2 Overall Performance

We first talk about the overall performance. In our experiments we select two baselines. The first one `linear-full-update` which simply extends SHORTLISTER by adding new parameters for new domains and conducting full model updating. The second `linear` is similar to the first baseline except that we freeze all trained parameters and only allow new parameter updating. Both the two baselines update the model with D_{k+1} dataset only. To show the effectiveness of each component of CONDA, we choose four variations. The first one is `cos` where we apply the Cosine Normalization (CosNorm). The second one `cos+der` applies CosNorm with the domain embedding regularization. The third one `cos+ns` uses both CosNorm and negative exemplars. And the last one `cos+der+ns` is the combination of all three techniques, which is our CONDA model. For hyperparameters, we pick $\Delta_{pos} = 0.5, \Delta_{neg} = 0.3, \Delta_{der} = 0.1, \lambda_{dsl} = 5$, and $\lambda_{norm} = 0.4$.

Figure 2 shows the accuracy for new domain adaptations. From the figure, here are the main observations. First, without any constraints, `linear-full-update` can easily overfits the new data to achieve 100% accuracy as shown in Figure 2(a), but it causes catastrophic forgetting such that the accuracy on seen domains is (almost) 0 as shown in Figure 2(b) and (c). By freezing the all trained parameters, the catastrophic forgetting

problem is a bit alleviated for `linear`, but the accuracy on the seen domains is still very low as we add more new domains. Second, `cos` produces much better accuracy on seen domains with a bit lower accuracy on each new domain, showing the effectiveness of the Cosine normalization. Third, as we add more regularizations to the model, we get better accuracy on the seen domains (Figure 2 (b) and (c)), at the cost of sacrificing a bit on the new domain accuracy (Figure 2 (a)). Also, `cos+der+ns` (the CoNDA model) achieves the best performance, with an average of 95.6% accuracy for each new domain and 88.2% accuracy for all previously seen domains after we add 100 new ones, which is only 3.6% lower than the upperbound (by retraining the model on the whole dataset). These demonstrate the superiority of our method.

4.3 Micro-benchmarks

Using Different Number of Initial Domains: We vary the number of domains for initial training to see if it will have a big impact on the model performance. Specifically, we pick 100 and 500 domains from InitTrain, and use the same IncTrain data for domain adaptation. Figure 3 compares the model performance on these three different number (i.e., 100, 500, 900) of initial training domains. From the figure we can see that the curves share a similar pattern regardless of the number of initial domains, showing that our model is stable to the number of domains used for initial training.

Varying the hinge loss thresholds: We vary the classification hinge loss thresholds Δ_{pos} and Δ_{neg} to see how it will affect the performance. Specifically, we fix $\Delta_{neg} = 0.3$ and vary Δ_{pos} from 0.5 to 1.0, and fix $\Delta_{pos} = 0.5$ and vary Δ_{neg} from 0 to 0.4, respectively. For both of the them we use 0.1 as the step size. Figure 4 shows the model performance. From the figures, we summarize the following observations. First, as we increase Δ_{pos}, on average the accuracy on each new domain gets better (Figure 4(a)), but we loss performance on all seen domains (Figure 4(b)). This is in accord with our intuition that a larger Δ_{pos} puts more constraint on the new domain predictions such that it tends to overfit the new data and exacerbates catastrophic forgetting on existing domains. Second, as we increase Δ_{neg}, on average the accuracy on each new domain gets worse (Figure 4(c)), but we get better performance on existing domains. This is because a larger Δ_{neg} narrows down the prediction "margin" between positive and negative domains (similar to decreasing Δ_{pos}), so that less constraint has been put onto predictions to alleviate overfitting on the new domain data.

Varying the domain similarity loss threshold: We vary the threshold Δ_{der} to see how it will affect the model performance. Specifically, we vary Δ_{der} from 0 to 0.5 with step size 0.1, and Figure 5 shows the model performance. As we increase Δ_{der}, the performance on the new domains gets worse, and the drop is significant when Δ_{der} is large. On the other hand, the accumulated accuracy on seen domains increases when we start to increase Δ_{der}, and drops when Δ_{der} is too large. This means we when we start to make the new domain embeddings to be similar to the existing ones, we alleviate the problem that the new domain dominates the domain summarization. Thus the accuracy on existing domains improves at the cost of sacrificing some accuracy on the new domains. However, if we continue to increase Δ_{der} to make it very similar to some of existing domains, the new domain will compete with some existing ones so that we loss accuracy on both new and existing domains.

Varying the weights for domain similarity loss: To see how the weighted domain similarity loss will affect the performance, we compare it against the plain version without the utterance similarity weights. Specifically, we set each $\lambda_i = \lambda_{dsl}$ having the same value. And our experiments show that the plain version gets the average accuracy 94.1% on the new domains, which is 1.5% lower than the weighted version, and 88.7% accumulated accuracy on all domains after adding 100 new domains, which is 0.5% higher than the weighted version. This means we can get a bit higher accumulated accuracy at the cost of sacrificing more new domain accuracy. In real applications, the decision to whether use weighted domain similarity loss should be made by trading off the importance of the new and existing domains.

Varying the number of used negative exemplars: As we mentioned before, we down-sample the negative exemplar set P to reduce the impact on new domain performance. To see if it's necessary, we compare it against the one without down-sampling. Our experiments show that without down-sampling, the model achieves

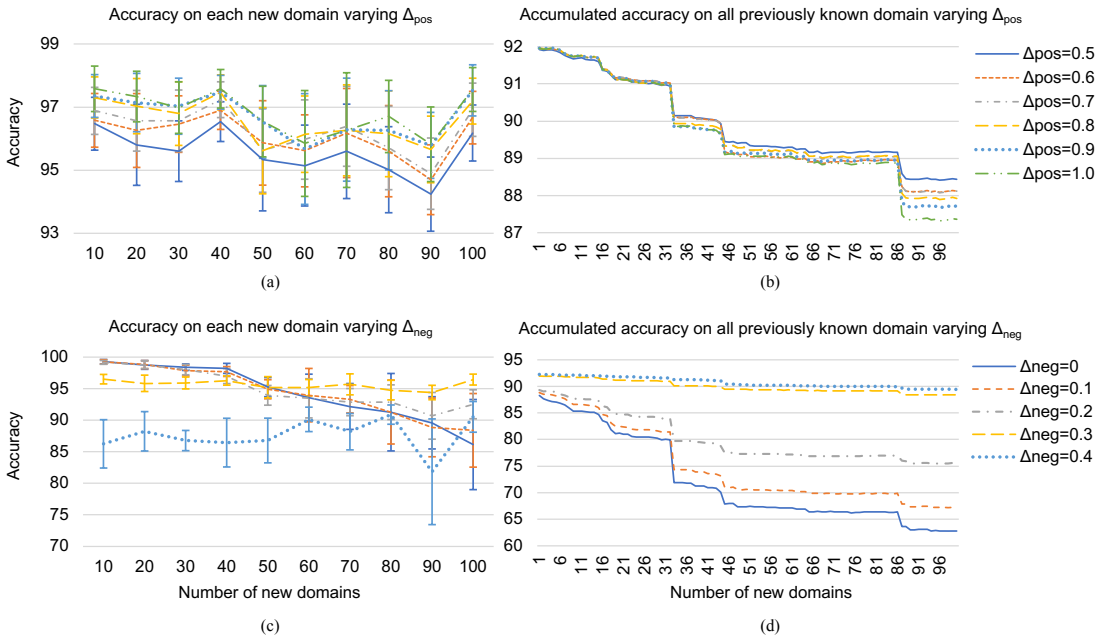

Figure 4: Model performance by varying the hinge loss thresholds. (a) and (b) shows accuracy on new domain and accumulated accuracy for all seen domains respectively by varying Δ_{pos}. Similarly, (c) and (d) shows the accuracy performance by varying Δ_{neg}.

Prediction Method	100 initial domains	500 initial domains	900 initial domains
Linear	95.2	93.9	93.4
Cosine	94.5	92.9	92.4

Table 1: Linear dot product versus Cosine normalization on initial training for different number of domains.

87.5% new domain accuracy on average which is 8.1% lower than the down-sampling version, and 87.2% accumulated accuracy on all domains which is 1.0% lower than the down-sampling one. This means down-sampling effectively improve the model performance.

Effect of Cosine normalization in initial training: We have shown Cosine normalization with hinge loss works better than linear dot product with sigmoid loss (used in SHORTLISTER) for CDA. Here we compare the two on the regular training setting where we train the model from scratch on a large dataset. Specifically, we compare the initial training performance on 100, 500, and 900 domains which are the same as we used earlier. Table 1 shows the accuracy numbers. From the table we see that Linear works better than Cosine by 0.7-1.0% across different number of domains. Though the difference is not large, this means Linear could be a better option than Cosine when we train the model from scratch.

Varying the order of the new domains: To see if the incoming order of the new domains will affect the performance, we generate two different orders apart from the one used in overall evaluation. The first one sorts the new domains on the number of utterances in the decreasing order, and the second in the increasing order. Denote these three orders as "random", "decreasing", and "increasing", and we conduct domain adaptation on these orders. Our experiments show that they achieve 95.6%, 95.5%, and 95.6% average accuracy on new domains respectively, and 88.2%, 88.2%, and 88.1% accumulated accuracy on all domains after accommodating all 100 new domains. This indicates that there is no obvious difference on model performance, and our model is insensitive to the order of the new domains.

Using more new domains: We also experimented with adding a large number of new domains to see the limit of CONDA. Figure 6 shows the results by continuously adapting 900 new domains one-by-one. From the figure we can see that at the early stage of the new domain adaptation (e.g., first 200 new domains), we get high new domain accuracy with little performance decrease on the existing domains. After that, the new domain performance becomes more unstable with violent oscillation, and the existing domain accuracy decreases more quickly. This suggests that we can-

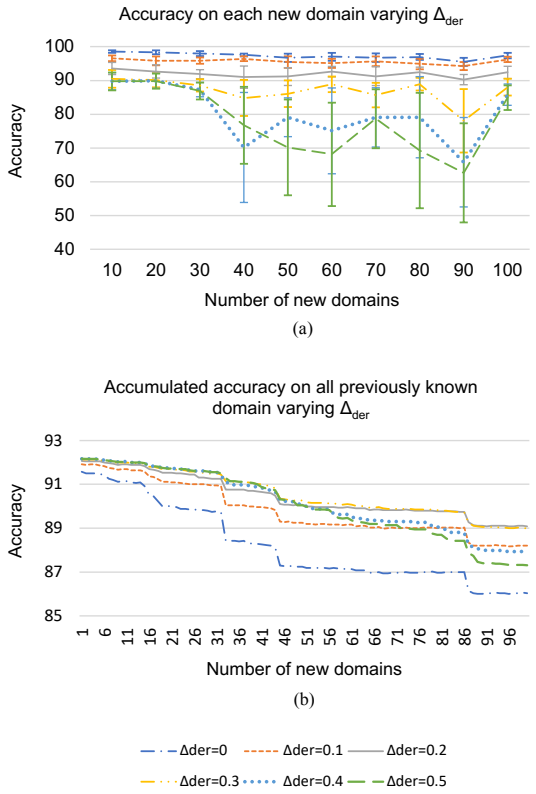

Figure 5: Model performance by varying Δ_{der}. (a) shows the accuracy on new domains, and (b) shows the accumulated accuracy for all seen domains.

not run the new domain adaptation forever, and after adapting a certain number of new domains (e.g., 200 new domains), it's more preferable to train the whole model from scratch.

5 Related Work

Domain Classification: Traditional domain classifiers were built on simple linear models such as Multinomial logistic regression or Support Vector Machines (Tur and De Mori, 2011). They were typically limited to a small number of domains which were designed by specialists to be well-separated. To support large-scale domain classification, (Kim et al., 2018b) proposed SHORTLISTER, a neural-based model. (Kim et al., 2018a) extended SHORTLISTER by using additional contextual information to rerank the predictions of SHORTLISTER. However, none of them can continuously accommodate new domains without full model retrains.

Continuous Domain Adaptation: To our knowledge, there is little work on the topic of continuous domain adaptation for NLU and IPDAs. (Kim et al., 2017) proposed an attention-based method for continuous domain adaptation, but it introduced a separate model for each domain and therefore is difficult to scale.

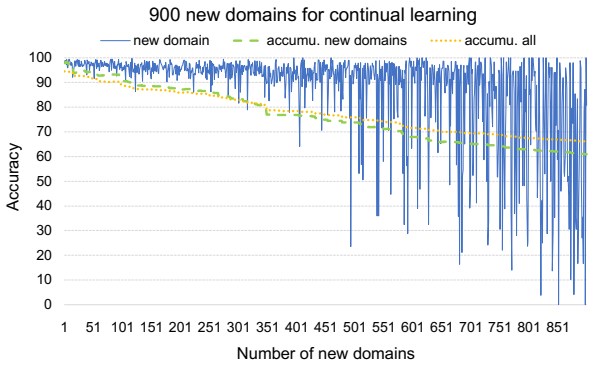

Figure 6: Using 900 new domains for continual learning.

Continual Learning: Several techniques have been proposed to mitigate the catastrophic forgetting (Kemker et al., 2018). Regularization methods add constraints to the network to prevent important parameters from changing too much (Kirkpatrick et al., 2017; Zenke et al., 2017). Ensemble methods alleviate catastrophic forgetting by explicitly or implicitly learning multiple classifiers and using them to make the final predictions (Dai et al., 2009; Ren et al., 2017; Fernando et al., 2017). Rehearsal methods use data from existing domains together with the new domain data being accommodated to mitigate the catastrophic forgetting (Robins, 1995; Draelos et al., 2017; Rebuffi et al., 2017). Dual-memory methods introduce new memory for handling the new domain data (Gepperth and Karaoguz, 2016). Among the existing techniques, our model is most related to the regularization methods. However, unlike existing work where the main goal is to regularize the learned parameters, we focus on regularizations on the newly added parameters. Our model also shares similar ideas to (Rebuffi et al., 2017) on the topic of negative exemplar sampling.

6 Conclusion and Future Work

In this paper, we propose CONDA for continuous domain adaptation. By using various normalization and regularizations, our model achieves high accuracy on both the accommodated new domains and the existing known domains, and outperforms the baselines by a large margin. For future work, we consider extending the model to handle unknown words. Also, we want to find a more principled way to down sample the negative exemplars.

References

Wenyuan Dai, Ou Jin, Gui-Rong Xue, Qiang Yang, and Yong Yu. 2009. Eigentransfer: a unified framework for transfer learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 193–200. ACM.

Timothy J Draelos, Nadine E Miner, Christopher C Lamb, Jonathan A Cox, Craig M Vineyard, Kristofor D Carlson, William M Severa, Conrad D James, and James B Aimone. 2017. Neurogenesis deep learning: Extending deep networks to accommodate new classes. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 526–533. IEEE.

Ali El-Kahky, Xiaohu Liu, Ruhi Sarikaya, Gokhan Tur, Dilek Hakkani-Tur, and Larry Heck. 2014. Extending domain coverage of language understanding systems via intent transfer between domains using knowledge graphs and search query click logs. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4067–4071. IEEE.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. 2017. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.

Alexander Gepperth and Cem Karaoguz. 2016. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation*, 8(5):924–934.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610.

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L. Hayes, and Christopher Kanan. 2018. Measuring catastrophic forgetting in neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*.

Young-Bum Kim, Dongchan Kim, Joo-Kyung Kim, and Ruhi Sarikaya. 2018a. A scalable neural shortlisting-reranking approach for large-scale domain classification in natural language understanding. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HTL 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 3 (Industry Papers)*.

Young-Bum Kim, Dongchan Kim, Anjishnu Kumar, and Ruhi Sarikaya. 2018b. Efficient large-scale neural domain classification with personalized attention. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2214–2224.

Young-Bum Kim, Karl Stratos, and Dongchan Kim. 2017. Domain attention with an ensemble of experts. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, page 201611835.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 971–980.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proc. CVPR*.

Boya Ren, Hongzhi Wang, Jianzhong Li, and Hong Gao. 2017. Life-long learning based on dynamic combination model. *Applied Soft Computing*, 56:398–404.

Anthony Robins. 1995. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146.

Ruhi Sarikaya. 2017. The technology behind personal digital assistants: An overview of the system architecture and key components. *IEEE Signal Processing Magazine*, 34(1):67–81.

Ruhi Sarikaya, Paul A Crook, Alex Marin, Minwoo Jeong, Jean-Philippe Robichaud, Asli Celikyilmaz, Young-Bum Kim, Alexandre Rochette, Omar Zia Khan, Xiaohu Liu, et al. 2016. An overview of end-to-end language understanding and dialog management for personal digital assistants. In *Spoken Language Technology Workshop (SLT), 2016 IEEE*, pages 391–397. IEEE.

Gokhan Tur and Renato De Mori. 2011. *Spoken language understanding: Systems for extracting semantic information from speech*. John Wiley & Sons.

Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual learning through synaptic intelligence. In *Proceedings of the 34th International*

Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, pages 3987–3995.