MetaLDC: Meta Learning of Low-Dimensional Computing Classifiers for Fast On-Device Adaption

Yejeta Liu
UC Riverside
yliu807@ucr.edu

Xiaolin Xu
Northeastern University
x.xu@northeastern.edu

Shijin Duan
Northeastern University
duan.s@northeastern.edu

Shaolei Ren
UC Riverside
sren@ece.ucr.edu

ABSTRACT

Fast model updates for unseen tasks on intelligent edge devices are crucial but also challenging due to the limited computational power. In this paper, we propose MetaLDC, which meta-trains brain-inspired ultra-efficient low-dimensional computing classifiers to enable fast adaptation on tiny devices with minimal computational costs. Concretely, during the meta-training stage, MetaLDC meta-trains a representation offline by explicitly taking into account that the final (binary) class layer will be fine-tuned for fast adaptation for unseen tasks on tiny devices; during the meta-testing stage, MetaLDC uses closed-form gradients of the loss function to enable fast adaptation of the class layer. Unlike traditional neural networks, MetaLDC is designed based on the emerging LDC framework to enable ultra-efficient on-device inference. Our experiments have demonstrated that compared to SOTA baselines, MetaLDC achieves higher accuracy, robustness against random bit errors, as well as cost-efficient hardware computation.

KEYWORDS

High-dimensional computing, low-dimensional computing, meta learning, fast adaption, scalability

ACM Reference Format:

1 INTRODUCTION

Deep neural networks (DNNs) have become the backbone of intelligent applications in a wide range of domains from computer vision to natural language processing [1, 5, 25]. Meanwhile, compared to cloud-based inference, on-device inference has numerous advantages, including better privacy preservation and anytime inference without relying on network connections. Nonetheless, despite the recent progress [27, 35], directly running DNN inference and adapting the model to unseen tasks on the edge are still challenging due to the conflict between high computational demand of DNNs and the low resource availability of edge devices, especially tiny devices such as microcontrollers and Internet-of-Things (IoT) devices.

In response to the excessive resource demand of DNNs, hyperdimensional computing (HDC) has emerged as an alternative towards efficient on-device inference [19]. The key idea of HDC is to encode data into (binary) hypervectors each with dimensions of thousands or even more, and then perform cosine/Hamming distance similarity for inference using bit-wise binary operations in parallel. Owning to its hardware friendliness and efficiency, HDC classifiers have been adopted to an increasingly broader range of inference tasks for resource-constrained devices [6, 10].

Nonetheless, there are still fundamental limitations that prohibit the applicability of HDC for tiny devices with extremely limited resources. First, the orders of megabyte of memory required by HDC to support its hyperdimensional data representation can be too costly for tiny devices [11, 18]. Second, compared to today’s DNNs, the HDC training process is extremely rudimentary (e.g., simply taking the average of inputs, plus some semi-blind heuristic adjustments, without loss functions), resulting in low inference accuracy. Last but not least, each HDC training process can only fit into one data distribution, which means that HDC training does not scale to a large number of tiny devices, each having potentially different distributions.

While the recent brain-inspired low-dimensional computing (LDC) classifiers outperform HDC by utilizing ultra-low dimensional vectors and principled training based on an equivalent neural network to improve the inference efficiency and accuracy [11], it cannot support fast model adaptation to unseen tasks on tiny devices. More concretely, LDC trains an individual model for each device, and hence the total training cost can be labor-intensive when there are many tiny devices deployed in heterogeneous environments each with different data distributions [24]. Additionally, fast adaption to unseen tasks with only a handful of data points presents substantial challenges for tiny devices, due to their constrained computing resources that prohibit traditional model updates based on gradients and backpropagation. While some studies have proposed to train a collaborative (DNN) model in a distributed manner to facilitate knowledge transfer between edge devices [24], the communication latency among edge nodes and complicated local neural network computation make this approach still too expensive for a tiny device.

In the presence of heterogeneous tiny devices each with a different data distribution, we propose MetaLDC, a new LDC-based meta learning approach that achieves fast model adaptation to an
we can obtain the encoded input sample $H$. In training, all $H$s belonging to the same class would be summed and averaged to obtain the class hypervectors, stored in the associative memory.

In the inference stage, the testing data would be transformed into the query hypervectors using the same encoder. Then a similarity checker like the Hamming distance would be applied in the associative memory between each trained class hypervectors and the query hypervector. The class label with the closest distance would be returned. Due to the simplicity of bitwise operations, the HDC has achieved success on platforms like FPGA and ASIC [6].

However, the large model size resulted from the ultra-high dimensions of the data representation in the HDC compromises its wide adoption on tiny devices, which are usually under severe resource consumption constraints. On the other hand, although numerous endeavors have been put into improving the accuracy of the HDC classifier [20], there is still a large accuracy gap between the HDC models and a simple modern neural network model like the Multi-Layer Perceptron (MLP).

**Low-Dimensional Computing (LDC):** To overcome the fundamental limitations of low accuracy and inference efficiency in HDC, the low-dimensional computing (LDC) classifiers are proposed as a brain-inspired substitute of HDC classifiers with higher accuracy and order-of-magnitude better on-device inference efficiency, especially for tiny devices with intelligent needs. Unlike HDC, LDC classifiers offer a systematic training procedure, where the value vectors $V$s and feature vectors $F$s are explicitly optimized rather than being randomly generated. On the other hand, the required order of magnitude of the involved vectors dimension size in the LDC is only a few to tens to achieve a higher accuracy compared to the state-of-the-art HDC, e.g., 87.38% w/ $D = 8,000$ vs. 91.22% w/ $D = 4,640$ on the MNIST dataset [11].

The blue and orange boxes with dashed lines in Figure 1 provides an overview of the LDC model architecture. The ValueBox is an encoding network. It maps the feature values $v$ into a bipolar feature values $V_{f}$, which is essentially a sparse binary neural network to bind the bipolar feature values $V_{f}$ with the corresponding feature vectors $F_{i}$ through the Hadamard product. The last class layer equivalently performs similarity checking, where the weights of the layer are collections of all class vectors. This layer outputs the score product for each class, and the class label with the highest score is taken as the classification result $\hat{y}$. It is worth noting that the inference in an LDC classifier is fully binary as the non-binary weights of ValueBox is not needed after training. Compared to HDC, LDC
classifiers have been demonstrated as a more promising alternative due to its lightweight model and high inference accuracy to support intelligent agents in the tiny devices.

3 PROBLEM SETUP

We focus on the few-shot supervised learning in this work. Supervised learning learns a model that maps input data points $x \in X$ which have a true label $y \in Y$ to predictions $\hat{y}$. A task $T_i$ is composed of $(X, Y, L, q)$, where $L$ is the task-specific loss function and $q$ is the data distribution of $T_i$. We assume all data points are drawn i.i.d. from $q$.

Given the distributions over a set of tasks $p(T)$, we aim at learning a general representation function $f(\cdot; \theta)$ using a handful of data points from each class. The $f(\cdot; \theta)$ is essentially a representation learning network parameterized by $\theta$, which can then fast adapt to previously unseen tasks in new devices by learning another adaption function $g(f(\theta); \phi)$ with little local data examples and (closed-form) gradient updates on $\phi$ with minimal computation. In a nutshell, we propose to train the LDC backbone as a reusable template to fast adapt to new tasks on tiny devices in the end.

With the goal of obtaining a good initialization, we train the LDC model in a meta-learning manner. Our MetaLDC consists of two phases: meta-training and meta-testing (also referred to as fast adaption). We meta-train on $m$ tasks $S_i \sim p(T)$, $i = 1, \ldots, m$ to learn the representation, and meta-test on a different task $T \sim p(T)$.

In the training process of MetaLDC, we introduce two updating loops as shown in our Algorithm 1. In a nutshell, the inner loop updates model parameters with respect to an individual task using $K$ data points by one (or few) gradient steps, while the outer loop updates the entire model’s parameters with regard to the loss after the inner loop updates.

In the fast adaption stage, we apply $M$-shot $N$-way evaluation, where $N$ is the number of classes per task. We would use $M$ data examples from each class of the new task to update the partial model, and then carry out the evaluation on the testing dataset from the new task.

4 THE DESIGN OF METALDC

In this section, we present our architecture, referred to as the MetaLDC. It uses a carefully-crafted interleaved training algorithm to train the LDC classifier. The primary objective is to achieve fast adaption to unseen (but related) tasks on edge tiny devices by updating partial learned model parameters with minimal computational cost. We provide an overview of the MetaLDC in Figure 1.

4.1 Meta Training

The original MAML [12] takes the training-and-fine-tuning pipeline, which optimizes $\theta$ and $\phi$ together in both meta-training and fast adaption. But, updating the non-binary weights $\theta$ can be too computationally expensive for tiny devices. Thus, in MetaLDC, we instead use the training-and-probing pipeline. The key idea of MetaLDC is to separate the representation function $f(\cdot; \theta)$ and the prediction function $g(\cdot; \phi)$ for meta-training and fast adaption on tiny devices. Thus, we only optimize the representation function $f(\cdot; \theta)$ in the meta-training, and optimize the prediction function $g(\cdot; \phi)$ in fast adaption.

Specifically, in the meta-training, for each update step, we first learn a randomly initialized prediction function $g(\cdot; \phi)$ to classify examples based on a given representation $f(\cdot; \theta)$. An updated parameter $\phi_i$ is obtained using $K$ examples from the sampled tasks $S_i$ through one (or more) gradient steps w.r.t. the loss on the sampled tasks. We then resample $K$ new examples from each class in $S_i$ and optimize the whole model w.r.t. $(\theta, \phi)$ across those tasks from $p(T)$.

The full algorithm is outlined in the Algorithm 1, where $\alpha, \beta$ represent tunable step size. Intuitively, the $\phi$ serves as the task-specific embedding, which modulates the behaviour of the model. In the outer loop updates, by considering how the errors on specific task changes with respect to the updated model parameters, we expect to obtain a model initialization such that small changes in the model parameters could lead to substantial performance improvement for any task. This improvement has been empirically attested in our evaluations of section 5.

Algorithm 1 MetaLDC—Training

**Input:** $T$: the whole task set; $t$: number of outer gradient steps; $m$: number of inner gradient steps (i.e., number of sampled meta-training tasks); $\alpha, \beta$: step size parameters.

**Output:** $\theta, \phi$

1: Randomly initialize parameters $\theta, \phi$  
2: for $j$ in 1, 2, ..., $t$ do  
3: for $i$ in 1, 2, ..., $m$ do  
4: Sample batches $B_i$  
5: Derive task-specific $\phi_i': \phi_i' \leftarrow \phi - \beta \nabla_{\phi_i} L(B_i; \theta, \phi)$  
6: Re-sample another batch $B_i'$ of the same batch size  
7: end for  
8: Update both parameters $\theta$, $\phi$: $(\theta, \phi) \leftarrow (\theta, \phi) - \alpha \nabla_{\theta} \frac{1}{m} \sum_{i=1}^{m} L(B_i'; \theta, \phi_i')$  
9: end for  
10: Return $\theta, \phi$

It is also worth noting that as the meta training involves potentially many tasks to learn a good initialization, the whole process of meta-training can be done on either the GPU or a powerful CPU. In contrast, adaptation is performed on local tiny devices with minimal computation cost as we explained in the subsequent subsections.

4.2 Fast Adaption

Given the learned initialization, we can fast adapt to a new task $T_i$ under a $M$-shot $N$-way setup. To adapt to a new task on each tiny device, we freeze the parameters of the representation network $\theta$ and only update the class layer’s parameter $\phi$ by using only a few samples. By this means, we preserve the broad knowledge learned from various tasks in the representation network $f(\cdot; \theta)$. Moreover, updating the last layer $\phi$ rather than the entire model is far less costly and therefore more affordable for tiny devices which are usually with stringent resource constraints. Furthermore, we use the hinge loss instead of the commonly used cross-entropy for gradient updates, as the former requires less complicated arithmetic operations. Instead of requiring the model to compute gradients by itself, we feed the gradients of the hinge loss in a closed-form to the model directly, via Eqn. (1):
5.1 Setup
We describe the datasets and tasks, baselines, as well as implementation details here.

Datasets. We use two benchmarking lightweight datasets, for both vision and non-vision applications, to evaluate MetaLDC on tiny devices. One is the Rotated MNIST, which is originated from MNIST [7]. In Rotated MNIST, each image contains the digit rotated by a certain degree. The other is referred to as Split ISOLET, derived from the UCI ISOLET dataset [31]. The ISOLET contains 26 classes in total, and we divide them into separate tasks, where each task contains 4 unique randomly sampled classes.

Training and Evaluation Tasks. In the Rotated MNIST, the rotation degree of tasks we use to train the methods are between \([0^\circ, 20^\circ, 40^\circ, 60^\circ, 80^\circ]\). The learned models are then evaluated on the testing dataset of Rotated MNIST with rotation degree from \([0^\circ, 20^\circ, 40^\circ, 60^\circ, 80^\circ]\). For the Split ISOLET, we use 20 classes from the ISOLET to generate the training tasks, while the remaining classes form the candidate classes pool to produce evaluation tasks.

Baselines. We compare MetaLDC with both HDC methods and other LDC-based models. In the Pretrained LDC method, we pre-train the LDC model with standard supervised learning using the whole training datasets from the training tasks. Then we also use the Algorithm 2 for fine-tuning. Another baseline we designed is the MetaLDC-full. The training algorithm of MetaLDC-full is the same as MetaLDC, as shown in the Algorithm 1. The difference is in the fine-tuning stage, where we update the parameters of the entire model, not only the last layer for the MetaLDC-full. Note that MetaLDC-full is prohibitively expensive for tiny devices, as it requires keeping the entire weights of the LDC model and full backpropagation calculations throughout the fast adaptation process. Besides the LDC variants, we also compare with a SOTA HDC method, the HDC with retraining. The HDC w/ retraining would give more weights to a misclassified sample in its correct class hypervector and subtracted from the wrong class hypervector in training to improve the HDC classification accuracy [18]. We set \(D = 8,000\) for the HDC models in our experiments following the setup of [11]. In addition, we put the multi-layer perceptrons (MLP) here as an upper bound, although it is not feasible to be deployed on extremely resource-constrained tiny devices. In the MLP, we use Algorithm 1 for training and Algorithm 2 for fast adaption.

Implementation Details. In the meta training stage, for the Rotated MNIST, we train the model for 60 epochs with a batch size of 10. In the Split ISOLET, we use 30 epochs w.r.t. its smaller dataset size. In the Rotated MNIST experiments, we used \(K \in \{1, 5\}\) for gradient updates in the meta learning involved methods, while setting \(K = 1\) in the Split ISOLET. We train all models with the Adam optimizer [22] except the HDC classifier, which is not feasible to fit in any standard training optimizer.

In the fast adaption stage, we set \(M = 10\) for the Rotated MNIST and \(M = 5\) in Split ISOLET, to update \(\phi\). To ensure fair comparison, the same data points are used across different methods. Note that we don’t fine-tune the HDC classifier as there seems no feasible way to update the learned hypervectors which are essentially composed of zeros and ones for the new incoming data examples.

5.2 Testing Accuracy
We show our evaluation results on the Rotated MNIST in Figure 2. From Figure 2, we can see that MetaLDC outperforms other baselines across all evaluation tasks. We observe that MetaLDC has achieved higher accuracy compared to the MetaLDC-full, which updates the entire model parameters to adapt to a new task in the fine-tuning stage. We attribute this to over-fitting, as the entire model focuses on learning a very small amount of data from the task. In comparison, MetaLDC, which only updates the last layer while keeping the former layers untouched, has alleviated this over-fitting issue to certain extent. We can also observe that as the rotation degree of the evaluation data becomes larger, the testing accuracy of the MetaLDC also increases due to higher similarity between the training and the evaluation tasks.
Table 1: Inference cost comparison between MetaLDC and the HDC w/ retraining on the Zynq UltraScale+.

<table>
<thead>
<tr>
<th>DataSet</th>
<th>Model</th>
<th>Size (KB)</th>
<th>Latency (us)</th>
<th>Energy (nj)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-MNIST</td>
<td>MetaLDC</td>
<td>6.48</td>
<td>3.99</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>HDC</td>
<td>1050</td>
<td>499</td>
<td>36926</td>
</tr>
<tr>
<td>S-ISOLET</td>
<td>MetaLDC</td>
<td>5.10</td>
<td>3.13</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>HDC</td>
<td>877</td>
<td>388</td>
<td>29488</td>
</tr>
</tbody>
</table>

Figure 3: Bit error robustness of different models on the MNIST testing dataset.

the highest on different tasks. The second highest is the MetaLDC-full, which is not as computationally efficient as the MetaLDC for tiny devices.

5.3 Robustness against Hardware Bit Errors

One of the most appreciated merits of the HDC-based models is the robustness against random bit errors on the hardware. The plain LDC model has been shown that it can achieve comparable robustness by the uniformly distributed information in the compact vector representation, given the largely reduced dimensionality [11]. Our empirical results in Figure 3 have shown that MetaLDC exhibits even stronger robustness than the pretrained-then-finetuned LDC model. This can be attributed to a more general representation learned via our meta training algorithm. The broad prior knowledge stored in the reusable template helps in adapting to tasks with noise perturbations and fighting against the random bit errors. Compared to the other methods, MetaLDC has also shown slower decline of accuracy.

5.4 Inference Cost on Hardware

Here, we evaluate the inference efficiency of MetaLDC and HDC w/ retraining following the same hardware pipeline setup as in the [11]. The hardware platform we use is the Zynq UltraScale+, where we transform the bipolar values {1, -1} to {0, 1} in the implementation. In the experiment, we limit the resource usage (e.g., lookup table (LUT) < 10k) to approach the common practice in tiny devices.

We report the numerical results in the Table 1. From the Table 1, we can see that the LDC models are at least 100x faster than HDC w/ retraining classifiers. The model size based on LDC is 150x smaller than the HDC ones. The energy consumption of the MAML LDC (tiny) are less than 100nj/ in the evaluation datasets, which has demonstrated great improvement on hardware acceleration compared to the HDC-based models. Note we don’t measure the MLP-based model cost here, as its inference requires matrix multiplication via floating-point operators rather than simple binary arithmetic, making it too resource-intensive to run on a tiny device.

On top of that, the MLP architecture cannot be trivially supported by the FPGA platform due to the floating point computation [13], and inter-platform comparison of algorithm performance is considered neither instructive nor fair. Even though there is MLP with the fix-point format which could be implemented on FPGA, the required utilization of DSP and other resources are still tremendous to carry out the involved matrix multiplication, far above the resource budget of a tiny device.
To study the efficacy of learned representation by our meta-training process, we conduct the ablation study for $K$, the number of data examples to train the model; and $M$, the number of samples we used from each class of a new task to update $\phi$ in the fast adaption stage. Depending on different dataset size, for the Rotated MNIST, we set the $K = \{1, 5, 10, 20, 40\}$, while $K = \{1, 2, 3, 4, 5\}$ for the Split-ISOLET dataset. The $M$ is set as $\{10, 50, 100, 150\}$ in the Rotated MNIST, and $\{1, 5, 10, 15\}$ in the Split-ISOLET, respectively. The evaluation dataset we used in the Rotation MNIST is the original MNIST, while the images in the training data have rotation degree in $\{10, 20\}$. In the Split-ISOLET, we use the Task 1 as the evaluation task.

The results are provided in Figure 4. We observe that as the values of $K$ and $M$ increase, the accuracy of both MetaLDC and MetaLDC-full has improved. We also notice that the accuracy gap between MetaLDC-full and MetaLDC become smaller as $K$ or $M$ increase, especially on the Split-ISOLET dataset. We attribute the performance increases of MetaLDC-full to larger percentage of data points sampled from the task to update the model, which gradually diminishes the over-fitting effect.

5.6 Effectiveness of the Learned Representation

To study the efficacy of learned representation by our meta-training process, we have designed another method, referred to as the MetaLDC-NonFineTuning (MetaLDC-NFT). In this method, we use the Algorithm 1 to train the LDC model. We then test its accuracy on the new task without any fine-tuning. As shown in the Figure 5, we can see that the accuracy gap between MetaLDC-NFT and MetaLDC is within 5% on the Split-ISOLET and 10% on the Rotated MNIST, which has reflected our Algorithm 1 has produced a good initialization to some extent.

6 RELATED WORKS

Meta learning. By distilling the learning experience from a broad set of related tasks, MAML [12] has achieved great success in the fast adaption regime [8, 14, 32, 36, 40]. The [24, 39] have proposed distributed collaborative frameworks to leverage knowledge between edge nodes via MAML. To reduce the computational cost, the [38] has presented a divide-and-conquer approach where the linear approximation is utilized to estimate the Hessian, while [30] has discussed using MAML with synthetic gradients in a feed-forward manner for deep neural networks. However, most approaches are still quite costly, not viable for tiny devices with severe resource constraints.

Hyper-dimensional computing. HDC has been known as an efficient alternative to expensive deep neural networks for tiny devices [3, 4, 9, 10, 15, 17, 29, 37]. The study [9] has proposed to use vector quantization to further reduce the model size. Besides, some works have optimized HDC’s encoding and training to improve its accuracy on a single data distribution [19]. Nonetheless, the required HDC model size to obtain an acceptable accuracy is still prohibitive large for tiny devices. More recently, LDC has been studied to significantly improve the efficiency of HDC [11], where the encoded vectors are only tens, yet the accuracy is even higher. Nevertheless, fast adaption problems to unseen but related tasks have not been well addressed in either HDC or LDC.

On-device learning. The fast adaption issue has become even more pressing for edge tiny devices due to their low latency tolerance and limited computational power [2, 26, 28, 33, 34]. For example, FSCL [21] and subsequent C-FSCIL [16] also utilize the frozen meta-learned network structure for adaption. However, they focus on addressing the catastrophic forgetting issue by storing the hyperdimensional prototypes of past classes in the online class incremental setting, rather than efficient edge device computing or adaption. Meanwhile, their replay buffer composed of hyperdimensional vectors is costly if deployed to tiny devices. In [23], the
MCUNet is proposed to find the optimal neural architecture by neural architecture search with resource constraints of heterogeneous tiny devices, whereas our MetaLDC does not require additional search efforts but provides a reusable lightweight template for unseen tasks.

7 Conclusion

In this paper, we propose MetaLDC, a LDC-based approach to fast adapt to unseen tasks via interleaved meta training for resource-constrained tiny devices. In MetaLDC, the LDC architecture is first trained across a set of different tasks, where we separately train the task-specific parameters $\phi$ in the inner loop of the meta-training algorithm. The learned model can then fast adapt to a new task by only updating the last layer using a handful of data points, while preserving learned prior knowledge in the former layers. Our empirical results have shown that our method has achieved higher accuracy compared to the HDC methods and pretrained LDC classifiers.

Limitations & Future Work

Our work has made the first step towards leveraging a novel low-dimensional classifier in on-device transfer learning for tiny gadgets and appliances. Further investigation on the feasibility of using other low complexity non-LDC NNs on tiny devices, as well as a more comprehensive comparison between them and the MetaLDC are desired in future works. Besides, extending MetaLDC to the unsupervised learning setting is a natural future direction to explore.

References


