Adaptive Risk Minimization

Ansh Kumar Sharma
Anunay Yadav
CSE618 - Meta Learning
Indraprastha Institute of Information Technology, Delhi

April 28, 2022

Introduction

Machine Learning Systems based on empirical risk minimization are designed keeping the assumption that the train and test data is drawn from the same underlying distribution. This assumption however is not taken care of most of the times.
Adaptive Risk Minimization (ARM) framework is designed to handle such cases with varying distributions. The paper uses the terminology of 'domain' as a proxy for distribution and the model is quick in adapting to different test domains with just unlabelled test data. In contrast to the popular paradigm of creating a robust model invariant across different domains, ARM is an adaptive model which is quick in learning to adapt to domain shift.
ARM like standard meta learning models is a 2-part framework - adaptation h and prediction model g.

$$h(\cdot, \cdot; \phi) : \theta \times X^K \longrightarrow \theta \tag{1}$$

$$g(\cdot; \theta) : X \longrightarrow y \tag{2}$$

Here, h works with k unlabelled datapoints and uses these to produce adapted prediction model parameters θ. g simply predicts a class for the unlabelled datapoints using the updated parameters.
The final optimization of ARM thus becomes,

$$\min_{\theta,\phi} \hat{\mathcal{E}}(\theta, \phi) = \mathbb{E}_{p_z} \left[\mathbb{E}_{p_{\mathbf{x}y|z}} \left[\frac{1}{K} \sum_{k=1}^{K} \ell(g(\mathbf{x}_k; \theta'), y_k) \right] \right] , \text{ where } \theta' = h(\theta, \mathbf{x}_1, \ldots, \mathbf{x}_K; \phi) .$$

The general framework for ARM looks as follows:

Algorithm 1 Meta-Learning for ARM

// Training procedure

Require: # training steps T, batch size K, learning rate η

Initialize: θ, ϕ

for $t = 1, \ldots, T$ **do**

  Sample z uniformly from training domains

  Sample $(\mathbf{x}_k, y_k) \sim p(\cdot\,, \cdot\,|z)$ for $k = 1, \ldots, K$

  $\theta' \leftarrow h(\theta, \mathbf{x}_1, \ldots, \mathbf{x}_K; \phi)$

  $(\theta, \phi) \leftarrow (\theta, \phi) - \eta \nabla_{(\theta,\phi)} \sum_{k=1}^{K} \ell(g(\mathbf{x}_k; \theta'), y_k)$

// Test time adaptation procedure

Require: θ, ϕ, test batch $\mathbf{x}_1, \ldots, \mathbf{x}_K$

$\theta' \leftarrow h(\theta, \mathbf{x}_1, \ldots, \mathbf{x}_K; \phi)$

$\hat{y}_k \leftarrow g(\mathbf{x}_k; \theta')$ for $k = 1, \ldots, K$

The paper proposes three variants of ARM:

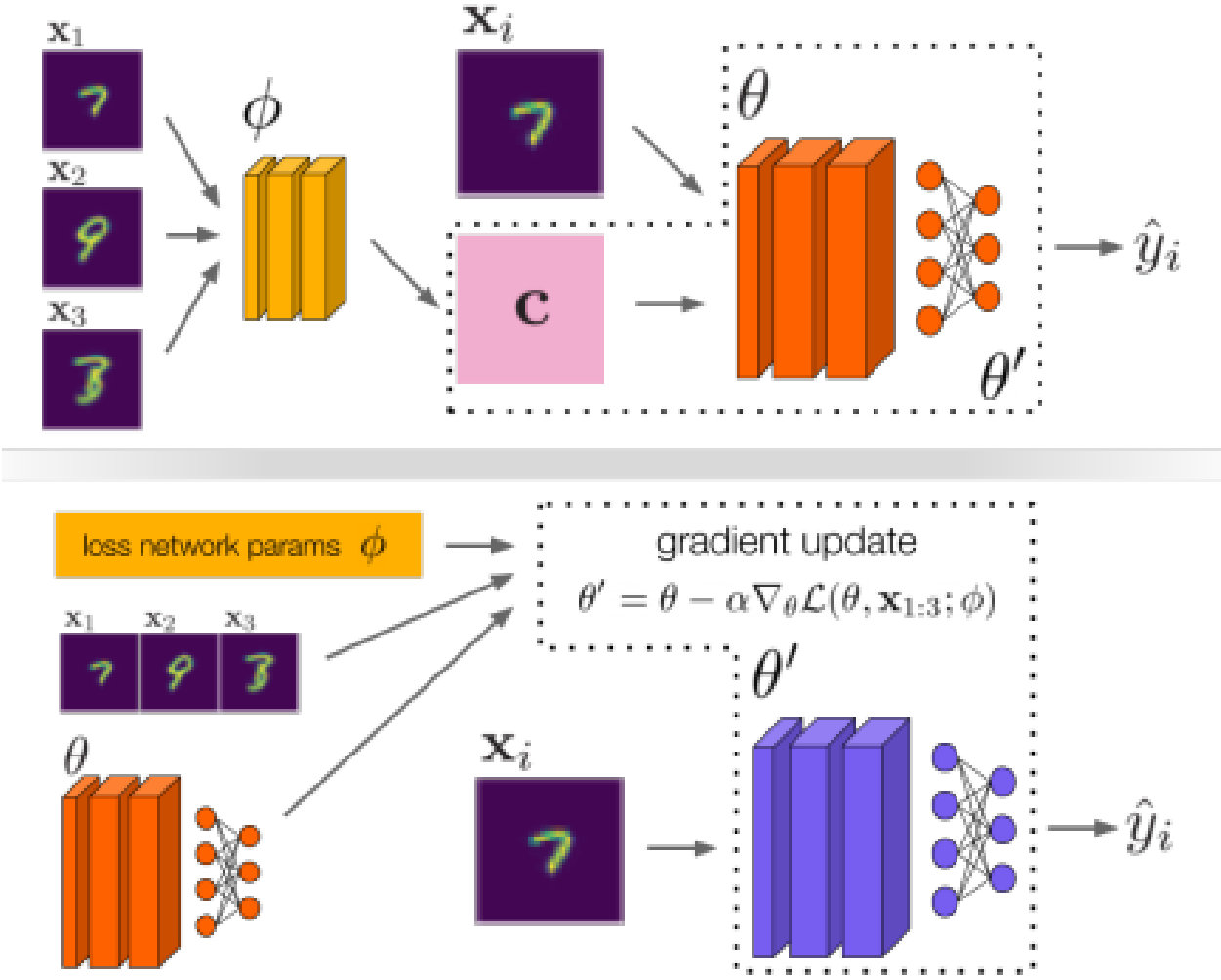
1. ARM-CML: h is a context network which learns a d-dimensional vector representation of of the unlabelled points. This context information is then concatenated with each input point and passed through the prediction network to get predictions.

2. ARM-BN: this model works with any g having batch-normalization layers. The two key aspects of ARM-BN are 1) training batches are sampled from a single domain, 2) normalization statistics are computed during test time.

3. ARM-LL: this follows optimization-based meta learning approach, unlike the other two. ARM-LL has a "loss network" f_{loss} which is used to compute scores for each prediction by the prediction model. This score is used to compute loss for backpropagation and thus updating the parameters of the prediction model. Here f_{loss} plays the role of h.

$$h(\theta, \mathbf{x}_1, \ldots, \mathbf{x}_K; \phi) = \theta - \alpha \nabla_\theta \|\mathbf{v}\|_2\,, \text{ where } \mathbf{v} = [\mathrm{f}_{\mathrm{loss}}(g(\mathbf{x}_1; \theta); \phi), \ldots, \mathrm{f}_{\mathrm{loss}}(g(\mathbf{x}_K; \theta); \phi)]$$

Adaptation model h for ARM-LL

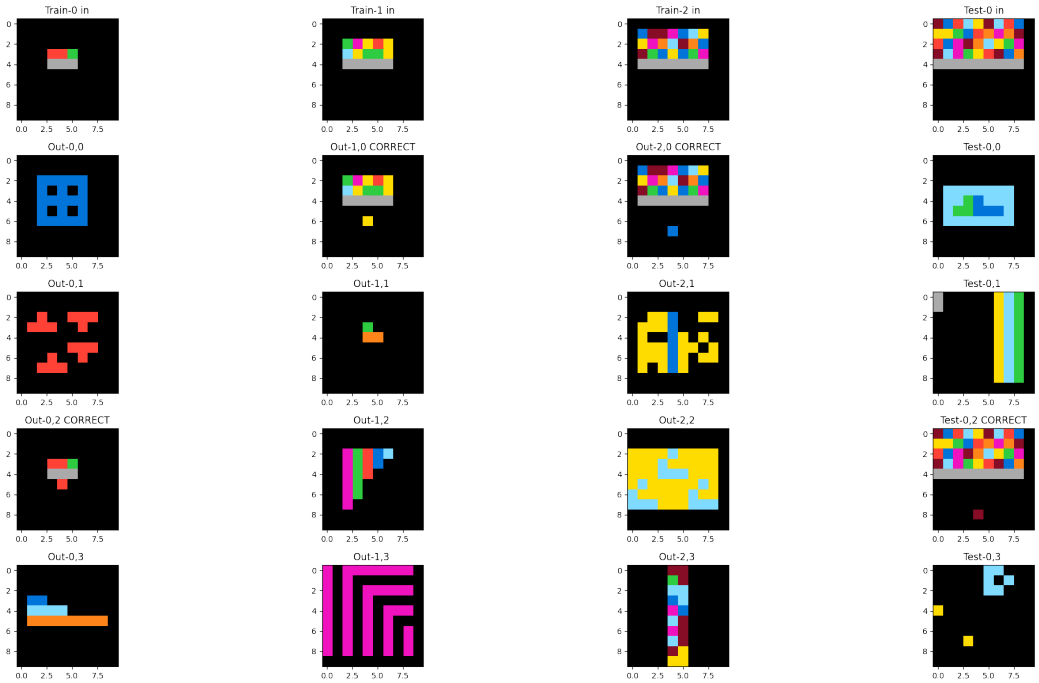


ARM-CML/BN (above) architecture compared with ARM-LL (below)

Datasets

ARC Dataset

ARC (Abstraction and Reasoning Challenge) comprises of the logical reasoning problems where you are given with one initial image and 4 future images, your task to choose the future image which most appropriately fits the previous image and the logic behind conversion from previous to future image. each task has its own logic which is the same for all the samples inside that task, but for each different task logic of converting initial image to future image is different hence this creates a meta learning task where we try to create a model which could adapt to the change in logic and predict the final image correctly. One example of the dataset is given below, where the logic is to extend blue cells in horizontally/vertically and pink cells diagonally



We converted the MCQ problem into a trainable dataset by combining each initial image with each of the options attached to it, 1 label was given to the pair which belongs to the correct answer and 0 otherwise. this creates a little skewness in the number of samples for each labels, as number of 0's = 3*number of 1s. but we can handle that by giving more weights to 1 label than 0 since, 1 label contains the information that we want to learn and 0 label contains noise in which there is no logic between the initial and the final image.

Market / Options Dataset

This dataset emulates the behaviour seen in current financial markets or options. We try to predict the stock price or classify whether the price of stock (etc) will go up or down. dataset comprises of total 5*25 tasks, where 5*20 tasks are meta train tasks and 5*5 tasks are meta test tasks. each task comprises of a training and testing dataset, Meta training comes into the problem as the data collected for each task is independent from others on several features such as time, etc. hence this creates a suitable meta learning problem for us to create a predictor which can easily adapt to different time events.

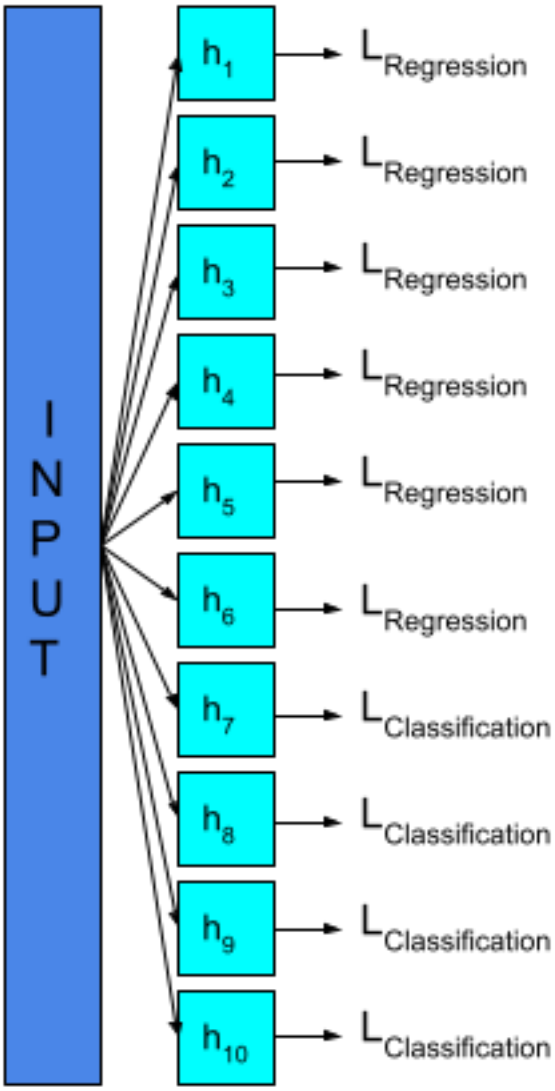
Market data: has both time series and flat dataset, each of which has an output of 10 dimensions. first 6 dimensions of the output(Y) are continous values and rest of them are labels.

Options data: has two different dataset "CE" and "PE", each of which is a classification problem into two classes. total dimension of each input is 148, and sample size varies from 32 to 19 size.

Dataset loaders of each of the above dataset are present in the Dataset folder with their respective names.

Methodology

We implemented the ARM variants and applied them to work on the three different datasets mentioned above. For both the prediction and adaptation model, we used simple MLP as our base model.
Since market data required 10 different tasks to solve, adaptation model had independent parameters for each task.



Adaptation model for market data task

We posed the adapting to domain shift problem for the given datasets by considering different tasks as separate domains. This enables us to apply ARM to meta learning datasets.

ARM-BN

ARM-BN is the batch normalized variant of ARM algorithms. We did two things specific to ARM-BN, 1) the training batches are sampled from a single domain, rather than from the

entire dataset, 2) the normalization statistics are recomputed at test time rather than using a training running average. g ARM-BN is pretty similar to ARM-CML hence in the paper they have considered both as the same contextual approach variants. for multi task learning like in market data, we used different models for each of the variable that we were trying to predict using the above mentioned adaptation model for market data task.

ARM-CML

for the implementation of ARM-CML, there were two different models involved in prediction, one called model-context which provides the model with the generalized context of the whole distribution which the model-predictions uses to solve the task. As stated in the figure ARM-CML(introduction). gradient flows through both the models starting from model-predictions to model-context. model context extracts important information out of all samples in a similar size representation and then all the context is combined by taking an average. for series data a combined vector was formed which was concatenated to each time embedding.

ARM-LL

The adaptation model in ARM-LL is called the loss network and it returns a scalar value for each prediction which is used to calculate loss. For implementing the inner optimiser to optimise during adapt, we used *higher* module by Facebook Research. In case of Market data, we created multiple optimizers - one for each task and thus had a separate loss network for each task. Since the weights of each task-specific model in prediction model were independent, this formulation made sense.

ARM-CNP

We formulated a variant of ARM, i.e. ARM-CNP which tries to learn the context model in a supervised setting. Class-wise mean vector of the output of the regular context model in ARM-CML is calculated and appended with the input. The combined vector is then fed to the prediction model. We try to find the question whether making the context learning supervised improve performance of ARM.

ARM-SUPLL

Another variant of ARM we formulated was the ARM-SUPLL which uses a supervised learned loss network. Along with the ouptut of the prediction model, we also append the true labels so that the loss network can learn some mapping using both input and output to finally give a loss. The rest of the working is similar to ARM-LL.

Experiments

Market - Series

	Task 1 (MSE)	Task 2 (MSE)	Task 3 (MSE)	Task 4 (MSE)	Task 5 (MSE)	Task 6 (MSE)	Task 7 (Acc)	Task 8 (Acc)	Task 9 (Acc)	Task 10 (Acc)
ARM_BN	516.42	**533.39**	778.48	838.77	748.61	502.13	0.282	0.295	0.281	**0.383**
ARM_CML	599.81	644.43	897.11	985.28	866.60	607.89	0.266	0.239	0.240	0.373
ARM_LL	**222.37**	576.62	**458.80**	**468.44**	**216.01**	**64.42**	**0.300**	**0.504**	**0.286**	0.255

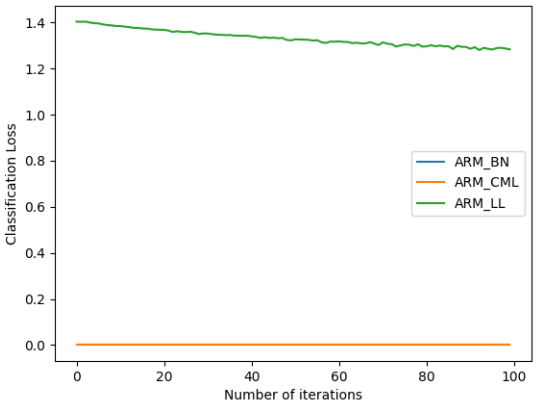
ARM-LL performed great by scoring highest in 8/10 tasks, but in general the MSE of all the tasks are significant implying that none of the model completely understood the problem and was able to solve it. for task 7 onwards, again similar thing happenend in all the tasks except task 8.

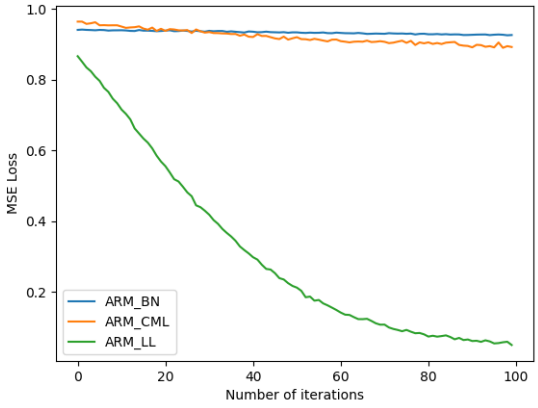
Market - Flat

	Task 1 (MSE)	Task 2 (MSE)	Task 3 (MSE)	Task 4 (MSE)	Task 5 (MSE)	Task 6 (MSE)	Task 7 (Acc)	Task 8 (Acc)	Task 9 (Acc)	Task 10 (Acc)
ARM_BN	230.68	**227.85**	177.63	361.29	418.0	**180.5**	0.327	0.768	**0.597**	0.304
ARM_CML	**13.10**	379.99	**1.74**	**2.01**	**2.42**	367.39	0.332	0.413	0.585	0.331
ARM_LL	147.69	305.04	190.43	199.23	182.97	317.53	**0.345**	**0.770**	0.546	**0.344**
MLP	227.68	200.74	186.55	355.30	397.83	214.31	0.29	0.493	0.502	0.317

Most of the models were not able to converge to a good point for all the MSE values except ARM-CML which was able to capture more information than others because of its contextual approach. MSE for task 1, task 3, task 4, task 5 were very low compared to other models. coming to classification ARM-CML was the second best model, but ARM-LL was the best model out of all. ARM-BN also got great results for task 9 because of the simplicity in the model prediction.

Loss plots for Market





ARM-LL takes heavy time in converging as it involves a gradient step before the prediction but in the classification tasks ARM-LL converged perfectly and achieved a whole new loss which the other models were not able to achieve.

Options and ARC

We have conducted extensive experimentation using the Options dataset as it gave more readable results - significant and robust.

	Options Dataset	ARC Dataset
ARM_BN	0.794	0.2803
ARM_CML	0.821	0.2403
ARM_LL	0.801	0.2441
ARM_CNP	0.891	0.2561
MLP	0.673	0.1832
MAML	0.912	**0.4031**
CNP	**0.931**	0.3604

As visible from the tabular results, standard meta-learning algorithms are capable of beating the ARM-based baselines by a significant margin. ARM-CNP is somewhat close for the Options dataset, but fails to perform for ARC.

Unrolling Steps

	Options Dataset
ARM_BN	0.794
ARM_CML	0.821
ARM_LL	0.801
ARM_LL_2	0.815
ARM_LL_5	0.832
ARM_LL_sup	0.843
ARM_CNP	**0.891**

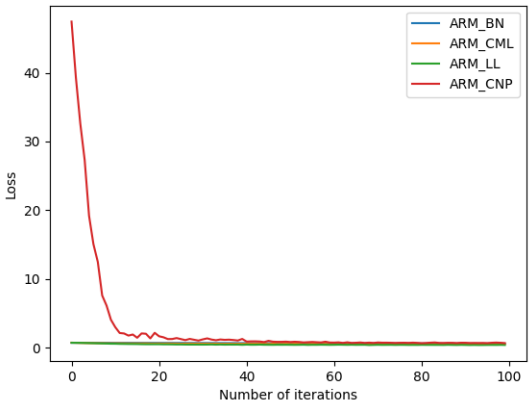
We performed experiment to see whether ARM-LL performance improved if we increased the number of inner updates before an outer update. As visible on the options dataset, the performance does improve with increasing unrolling steps. Convention: ARM_LL_i where i denotes number of unrolling steps. ARM-LL-sup is able to beat all the existing variants except the ARM-CNP, because it is learning loss in a supervised manner.

Overlap between test and train domains

	Options Dataset (Non-overlap)	Options Dataset (Overlap)
ARM_BN	0.794	0.819
ARM_CML	0.821	0.832
ARM_LL	0.801	0.823
ARM_CNP	**0.891**	**0.847**

We also tested the performance of our models when working with overlapping domains between test and train set. We got intuitive results with an increase in performance in most cases when the domains overlapped.

loss plot for Options data



ARM-CNP is a more complex algorithm hence takes more iterations to converge than other algorithms. Other algorithms start off with a great accuracy.

Contributions

The key components of our project are as follows:

1. Detailed study of the paper and relevant work.
2. Implementation of the ARM model along with its three variants.
3. Extending ARM to the given three datasets.
4. Formulating and implementing two additional variants.
5. Running experiments including change in number of unrolling steps, overlap between test and train tasks and analysis of losses and results.