Proceedings of the 19th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 157–167
DOI: 10.15439/2024F4426
ISSN 2300-5963 ACSIS, Vol. 39

Model-Agnostic Machine Learning Model Updating – A Case Study on a real-world Application

Julia Poray
0009-0003-4868-1255
GlobalFoundries Dresden Module One LLC & Co. KG
julia.poray@globalfoundries.com

Bogdan Franczyk*
0000-0002-5740-2946
Leipzig University, Information Systems Institute
Grimmaische Straße 12, 04109 Leipzig
franczyk@wifa.uni-leipzig.de

Thomas Heller*
0009-0006-9221-3784
GlobalFoundries Dresden Module One LLC & Co. KG
thomas.heller@globalfoundries.com

***Abstract*—The application of developments in the real world is the final aim of all scientific works. In the case of Data Science and Machine Learning, this means there are additional tasks to care about, compared to the rather academic part of "just" building a model based on the available data. In the well accepted Cross Industry Standard for Data Mining (CRISP-DM), one of these tasks is the maintenance of the deployed application. This task can be of extreme importance, since in real-world applications the model performance often decreases over time, usually due to Concept Drift. This directly leads to the need to adapt/update the used Machine Learning model. In this work, available model-agnostic model update methods are evaluated on a real-world industry application, here Virtual Metrology in semiconductor fabrication. The results show that for the real-world use case sliding window techniques performed best. The models used in the experiments were an XGBoost and Neural Network. For the Neural Network, Model-Agnostic Meta-Learning and Learning to learn by Gradient Descent by Gradient Descent were applied as update techniques (among others) and did not show any improvement compared to the baseline of not updating the Neural Network. The implementation of the update techniques was validated on an artificial use case for which they worked well.**

***Index Terms*—Machine Learning, Concept Drift Adaptation, Model Updating, Supervised Regression, Model degradation, Semiconductor Fabrication, Virtual Metrology**

I. Introduction

In modern industrial environments, e.g. semiconductor fabrication, there is an increasing use of Machine Learning models throughout the whole processing chain and beyond. A high amount of use potential lies in Machine Learning regression tasks which aim to provide hard to obtain quality parameters. One such case is Virtual Metrology [1][2], where measurement data are predicted from process machine information, i.e. sensor data and more. An example for such a measurement is the thickness of a layer deposited on a wafer. Data from such an example are used in this work.

However, there are numerous other use cases as well. Given the dynamic nature of the industrial environment, a model once deployed requires an update after some time.[1] The reason for that is mainly Concept Drift [4], which may be summarized as a change of the response surface of the underlying problem over time [5]. Although Concept Drift is rather uncommon in pure academical tasks, it is of major importance in certain real-world applications. To emphasize that: in semiconductor fabrication, it is one main task for countless process and equipment experts to react on drifting processes or process machines.

A paper that garnered considerable attention was published by Gama et al. in 2014 [6], introducing a taxonomy for the subject area which is used in this article. This was followed by additional works summarizing various techniques for adapting to concept drift: In their 2019 review on Concept Drift Adaptation (CDA), Lu (and Gama) et al. analyzed 130 papers that included adaptation techniques [7]. They discovered that there are techniques which utilize retraining with varying window sizes, ensemble techniques used for adaptive purposes, and model-specific techniques like adaptive decision tree models. Another survey on Machine Learning for recurrent concept drift was conducted by Choudhary et. al. [8]. They listed additional methods that can be grouped into 1) The use of strategies for selecting, weighting or multiplying specific data points for retraining [9] [10] [11] and 2) Dynamics through adaptation in preprocessing [12], [13]. Suárez-Cetrulo et. al. additionally list Meta-learning techniques in the context of Concept Drift adaptation [5], p.9 ff. Meta-Learning has also been a topic that has continued to gain attention. Vanschoren [14] provides an overview of the Meta-Learning concept and its application areas. Among other Meta-Learning applications, they deal with Meta-models that "recommend the most useful [algorithm] configurations [...] given the Meta-features [...] of a task [...]" ([14], p.10). One option to do this is Transfer Learning ([14], p.12). With this, optimized models from previous tasks are taken as a warm-start to be trained on

*These authors contributed equally to this work and are listed in alphabetical order.

[1]Model updates are part of the maintenance phase in context of the Cross Industry Standard for Data Mining (CRISP-DM [3]).

Topical area: Advanced Artificial Intelligence in Applications

new tasks and yield a new model optimized for the new task.

The intersection between Meta-Learning and Concept Drift adaptation was reviewed in a survey published in 2023 by Son et. al. [15]. It contains several Meta-Learning techniques primarily used for the adaptation of Deep Learning models. In this development, it is clear to see that the topic of model adaptation has received a lot of attention and there have been continued efforts put into developing sophisticated techniques for model adaptation (for example [16], [17], [18], [19]).

In summary:
For a real-world deployment of a Machine Learning model that suffers from Concept Drift, there has to be a model update implementation to keep the application usable.
Consequently, there is a demand for techniques that can effectively update regression models. Furthermore, when multiple update techniques are available, it becomes important to determine which technique is most suitable for a specific use case and model type.

To qualitatively understand the term “most suitable”, two constraints are taken into account: 1) the required model accuracy, and 2) the availability of new labeled data.

The question then arises:
Which update technique extracts information from the new labeled data and integrates them into the model in the most efficient way?

While the inclusion of the accuracy requirement is obvious, the availability of new labeled data may need more clarification. Since new labels are expensive to obtain, they usually are not available in a high (enough) frequency. An example in the presented use case from Virtual Metrology are maintenances of the process machine: it is not uncommon that the whole distribution of a feature value jumps to another unknown(!) distribution mean due to the maintenance. So, the model’s predictions after machine maintenance will not be accurate enough. Since the frequency, of which new labeled data is available, is low, the data collection for a model update takes long. Hence, the aim is to adapt the model to fit the accuracy requirements with as few new data points as possible.

It should be stated clearly that this work does neither focus on the origin of the Concept Drift, nor what the Concept Drift looks like. Only the adaption of the model to fit the accuracy requirements is regarded.

There is one additional question that shall be addressed: Why are only model-agnostic update techniques regarded in this work?

The answer may be a bit uncomfortable: In the underlying industrial environment, the utilization of very specific techniques, e.g. a special type of Machine Learning Model and corresponding to that, a special model update technique, scales badly. It is much more beneficial to be flexible regarding the used Machine Learning model types. Especially, when the code bases of Machine Learning libraries get updated or completely new model types are developed, the deployed model update setup should still work.

Given the multitude of potential use cases and the variety of model types, one would want to apply update techniques that can be scaled to different model types. It eliminates the need to start from scratch with each model, thereby making it economically more viable[2].

There are also other circumstances that lead to additional requirements on the techniques used: Typically, there is a scarcity of historical data for a given use case or process. This is primarily due to the high cost associated with retrieving labeled data, such as physical measurements, which are the labels for the supervised learning task in case of Virtual Metrology. Moreover, this work posits that the update techniques should also be applicable to newly introduced use cases or models which solve similar underlying problems. The idea is to let the model reuse already learned knowledge from the old case and relearn similar patterns again just based on the new case’s data.

This paper compares existing techniques to update regression models by applying them to a real-world use case from semiconductor fabrication (deposition of a layer on a wafer). They are also applied to an artificial use case to compare the behavior of the techniques in a setting where no unknown effects are present. As this work focuses on supervised regression tasks, the techniques considered here should be developed for regression tasks or be straightforwardly applicable to them. Extending update techniques from classification models to regression models would be a topic for "future work." With the learnings in this work, a starting point for scaling update techniques across use cases with different model types in fabrication is provided.

The remainder of this article is structured as follows: section I-A provides an overview of the related work. The setup of the experiments is explained in section II. There it is also listed which update techniques are used. Section III presents the results with discussion. A summary and conclusion is given in section IV.

A. Related Work

A range of adaptive models can be found in literature. Examples include Adaptive Random Forest [20], Adaptive XGBoost for classification [21] and regression [22], and Adaptive Support Vector Machines [23] and others [24]. Neural Networks are inherently incremental learners, and there are works that introduce structures to Neural Networks to handle Concept Drift. For instance, Memory-Augmented Neural Networks [25] and Adaptive Extreme Learning Machines [26] for classification and [27], to name a few.

Why not use adaptive models? The choice of model type is specific to the use case. From experience in modeling processes in fabrication, Deep Learning models are often not utilized, primarily due to insufficient amounts of data to train them (costly data). In the context of this paper’s objective, which is to gain insights into model adaptation techniques that may be transferable to other use cases, it is sensible to investigate techniques applicable to any model type.

Update techniques that can be applied to any model type may be categorized, in the authors’ view, into 1) Selection

[2]Data Science resources are limited

and weighting of data points given to the model for updating (whatever mechanism the model then uses to update itself) [28], [29]. 2) External optimizers – that is, the internal model parameters are set to new values by an external optimizer and then reassigned to the model [30], [18], [19]. 3) Ensemble Methods [31].

The three cases are discussed further in the following:

1) Selection and weighting of data points for updating: Rolling Window [6] involves different window sizes and weighting of the input data in the loss function. This can be done linearly or exponentially decreasing against the age of the data. There are also newer techniques for determining the weights of the data points, or even leaving out data points entirely. For instance, [32] uses fuzzy models to select which data are used. Similarly, [33] employs fuzzy kernel c-means clustering of the data stream to determine the data points for the update. This is combined with a forgetting policy during the update. The techniques included in this paper should be applicable with as little lead time and thus as little historical data as possible. Given that it is assumed that there will generally not be enough historical data available for the auxiliary models for selecting update data for the regression model, these techniques are not applied here. The same applies to techniques that store a "model history" and make a selection with various strategies as to which model or which composition of historical models should be used for new inference data.

2) External optimizers: What can an external optimizer do better than a model's own one? Possibly, information from past data can be stored in an external optimizer. This happens in Meta-Learning. An example of this is Andrychowicz' technique "Learning to Learn by Gradient Descent by Gradient Descent"[3] to optimize an algorithm using a Meta Learner [30]. As optimizer (in this case called Meta-Learner), they use Long Short-Term Memory (LSTM) networks [34]. The LSTM's task is to predict the changes in the algorithm's parameters given the gradient of the loss of the task of the algorithm with respect to the algorithm's parameters. This means, the LSTM predicts the changes in the Neural Network's internal parameters (weights and biases) given the regression/classification loss' gradient with respect to the weights and biases of the Neural Network. In their paper, the training of Neural Networks on the MNIST [35] and CIFAR-10 [36] dataset are used to demonstrate the developed architecture. Li and Malik [19] developed a similar approach, formulating the problem in a reinforcement learning setting.

Another Meta-Learning approach is Model-Agnostic Meta-Learning (MAML) [37]. Here, starting values are sought for the internal model parameters, from which the best accuracy is achieved when using a fixed optimizer to do the adaptation to new data. The key idea behind MAML is to find a model initialization that is not only good for one task but can be quickly fine-tuned for any task in the distribution of tasks. During the training phase, MAML performs a two-level optimization process. In the inner loop, it learns a separate model for each task using the inner optimizer. For example, the optimizer can use several gradient steps. In the outer loop, MAML updates the initial model parameters based on how well the task-specific models performed. The goal is to find a set of initial parameters that, when fine-tuned on a few examples from a new task, can achieve good performance on that task. Finn et. al. [37] demonstrate their technique on Neural Networks solving an artificial regression use case (sine wave) and classification tasks.

The Meta-Learning techniques mentioned above were applied in the Deep Learning field [15]. The techniques use gradients of the loss function wrt. internal model parameters in the optimization processes [30], [37], [18]. The name of the technique "Learning to Learn without Gradient Descent by Gradient Descent" [38] may suggest that computing the gradients of the loss function with respect to the internal parameters of the model to be optimized is omitted. Indeed, the gradients are not fed into the Meta Learner during inference phase. But to train the Meta Learner, still, the gradients mentioned are necessary.

For Neural Networks or much simpler multilinear regressors, for example, these techniques are straightforwardly applicable. Theoretically, the application of the techniques or ideas could also be extended to models with discontinuities in the loss function by finding ways to artificially make the loss function continuous (e.g., overlaying of a smoothing function) or other numerical calculation of the gradients. Extending the techniques that work well in the Deep Learning field to other model types such as Trees / Forests / GXBoost is an interesting task. However, this is a separate work. As an example, two of these techniques are nevertheless applied to the Neural Network in this article.

3) Ensemble techniques. In the context of ensemble techniques, there exists the utilization of multiple model types and the (weighted) aggregation of various results [39],[40],[41],[42]. The construction of different models based on diverse input data and the subsequent aggregation of results is referred to as Bagging, with Adaptive Bagging also being a notable variant [31]. An advanced approach for the latter two involves predicting the weights with a meta-model. However, this approach is not adopted here due to the general lack of sufficient training data at the onset. Another ensemble technique is Boosting, where the next model is fitted on the residuals.

This paper applies a selection of model update techniques to a real-world regression task from a use case in semiconductor fabrication and compares these update techniques, which are practically applicable. The selection of model update techniques is based on the industry requirements discussed in Section I. Furthermore, it is a requirement that the techniques can be applied as best as possible to newly introduced regression models.

The literature provides two studies on the application and comparison of update techniques:

Celik et. al. investigate update strategies for Auto-ML

[3]This update technique is referred to as "grad2-lstm" in this article.

systems in the face of Concept Drift [43]. The strategies employed encompass various configurations of “Detect and Restart” vs. “Periodic Restart” and “warm-start” vs. “retrain” vs. “AutoML-Restart”. The difference between “retrain” and “AutoML-Restart” is that “AutoML-Restart” includes retuning of hyperparameters. A not unexpected result, quote [43]: “different drift characteristics affect learning algorithms in different ways, and that different adaptation strategies may be needed to optimally deal with them”. This indicates that there is no one-for-all technique regarding model adaptation.

The work at hand does not focus on Auto-ML systems but looks at a level below, to see the direct effect of different update strategies.

In their benchmark on “Learning to Optimize” (LTO), Chen et. al. [44] compare optimization techniques on various benchmark tasks. One of these tasks is the training of Neural Networks. The concrete optimization tasks are training a Multilayer Perceptron and a Convolutional Neural Network with different optimizing techniques on MNIST data set [35]. For both experiments, as traditional optimizers, Adaptive Moment Estimation (ADAM) [45], SGD [46] and Root Mean Square Propagation [47] are used. Four Learning-to-optimize approaches are used in the experiments [30], [48], [49], [50]. For all of them, the architecture of the optimizer contains recurrent Neural Networks such as LSTM architectures. Among these is the approach from Andrychowicz et. al. [30], which is used in the work at hand as well. The findings of Chen et. al. for the experiments of training Neural Networks are "lacking stability during testing" ([44], p.35) for all of the LTO-optimizers whereas the traditional optimizers show convergence when training each Neural Network. Training the Multilayer Perceptron, the optimizer from [30] diverges. It works for training the Convolutional Neural Network, but only for a small number of iterations. Two of the other three LTO-approaches show good results on the Multilayer Perceptron but can not do efficient optimization on the Convolutional Neural Network [48], [49]. The use of an advanced training scheme for LTO-Optimizers [50] allowed [44] to improve the optimizing performance on both Neural Network optimizees.

It is worthy to mention that the two experiments from [44] were carried out on Neural Network optimizers that performed classification of the MNIST [35] dataset. The article at hand aims at ongoing optimization of models for regression tasks (which can be but are not necessarily Neural Networks). For our experiment with a Neural Network performing regression tasks, [30] is used as a first benchmark technique on the given regression data.

To the best knowledge of the authors, there is no comparison work yet that focuses on model type independent update techniques and compares their functionality on different model types.

Especially, a comparison of different update techniques on different regression models (model types) on the same real-world regression task is important but not available. This paper makes a starting point for closing this gap. The goal is to gain knowledge about which update techniques perform better on which model type in an industrial regression setting.

II. Setup of the Comparison

To evaluate the performance of the different model update techniques, multiple experiments are conducted, where a model is trained on a given data set and afterwards used with drifting test data. During the test, the model is continually updated with latest test data using the different update techniques. Two independent cases are regarded: the first one is a real-world application from an industrial environment and the second one is based on artificial data to get an undisturbed view on the update techniques.

A. Real-World Use Case

The industrial use case is Virtual Metrology of a layer deposition in semiconductor fabrication. The regression data set consists of 32 features (numerical and encoded categorical) and one label. A feature selection process was done before this. The label (and prediction) data are the thicknesses of a layer deposited on a wafer and are scaled to maintain confidentiality.

For the Virtual Metrology use case, the available labeled data was divided into $165,000$ data points for training and $250,000$ data points for the comparison of the update techniques. The data points for the comparison were divided into batches of 50 data points yielding 5000 batches. Two models were trained including hyper parameter optimization: An XGBoost model and an Artificial Neural Network[4] ([51], [52]). After the training, batches of feature data points are fed to the models in historically correct order and compared to the corresponding label data. One batch contains 50 wafers (data points). Each batch is handled as follows:

A train-test-split is made yielding $0.6 * 50 = 30$ wafers for updating and $0.4*50 = 20$ wafers for evaluation of the updated model. The latter provides the accuracy for the given batch. As accuracy measure, the mean absolute error (MAE) is taken. For each update technique, for each batch, the accuracy is stored. To make a statistically valid statement about the accuracy, over the stored accuracies, the mean of the accuracies in the window is calculated, using a rolling window of size 500.

The question asked is which update techniques work how well on different model types. This is assessed by using two setups: blind adaptation and informed adaptation [6]. Blind adaptation means the model is updated regularly without a trigger, whereas informed adaptation means the model is updated when triggered, e.g. via a Concept Drift detector or a performance degradation detector. For the blind adaptation setting, the model is updated with the corresponding update technique regularly in intervals of 200 batches. The frequency of updating and the update batch size were not varied because it is not the focus of this work. As loss function the mean absolute error is used.

[4]Sequential, dense Neural Network with the structure: Input Layer of size 32, Dense Layers with [64,128,64,64,32,32,1] nodes. Each layer is followed by a ReLU activation function.

For the informed updating case, in this work, a performance degradation detector is used: the model accuracy (here: the model loss on available new labeled data) exceeding a fixed threshold. In practice, this threshold is use-case specific. In this study, on the same real world use case, the experiments are conducted with two different model types: XGBoost and Neural Network. The accuracy of the initially trained XGBoost model is higher than the initial accuracy of the Neural Network. In practice, for this use case, the XGBoost model would be chosen. For academic purposes, possibly allowing to gain knowledge for other use cases, in which a Neural Network might achieve better performance, the experiments are conducted on both model types. The threshold used for the performance degradation detector is oriented at the initial accuracy of the corresponding model for the use case, leading to a higher threshold for the Neural Network than for the XGBoost model. It is important to mention that in practice, there would only be one threshold per use case. If a model would exceed this threshold in the beginning, it would not be, and is not, chosen.

To maintain confidentiality, neither the absolute values of the model performances nor the threshold values are given. The loss values, shown in the III section, are scaled to relative units.

Quantification, which update technique performed best, was done by calculating the overall mean loss over the whole 5000 batches.
The results are shown and discussed in Section III.

B. Generation of the Artificial Example Data

To validate the implementation of the adaptation techniques without any unknown influences, artificial training data were constructed. They consist of five numerical input data where each input lies in the interval $[0, 1]$. Independently, uniformly randomly sampling each variable from this interval, one corresponding numerical output (label) is generated using a multilinear function with known, fixed parameters a_1, a_2, a_3, a_4, a_5 and b:

$$f : [0,1]^5 \longrightarrow \mathbb{R} \tag{1}$$

$$(x_1, x_2, x_3, x_4, x_5) \rightarrow f(x_1, x_2, x_3, x_4, x_5) \tag{2}$$

$$f(x_1, x_2, x_3, x_4, x_5) \coloneqq \sum_{i=1}^{5} a_i \cdot x_i + b \tag{3}$$

$$a_i \in \mathbb{R} \text{ constant for } i \in \{1, 2, 3, 4, 5\} \tag{4}$$

For the concrete example used here, $a_1 = 1.0$, $a_2 = 0.5$, $a_3 = -0.1$, $a_4 = -1.5$, $a_5 = 0.005$ and $b = 0.7$ were chosen for the pretraining data generation. The pretraining data consist of 200 batches ($10,000$ data points). For the comparison of the update techniques during inference, a data set – similar to the training data – is generated, which includes Concept Drift. The inference data consist of 20 batches containing 50 data points each. The inference data generation is done by changing the parameters a_3 and b three times, every fifth, batch, in the following way: The first five batches of the inference data were generated exactly from the same function as the pretraining data. The data in the sixth batch are generated in the same way but changing the parameters in function (4) to

$$a_3' = a_3 + \Delta a_3, \quad \Delta a_3 = 0.2, \quad b' = b + \Delta b, \quad \Delta b = 0.2.$$

Five batches of the new configuration were appended to the inference data. Then again, starting from this parameter configuration data generating function (equation 4), the function parameters are shifted again by $\Delta a_3 = 0.2$ and $\Delta b = 0.2$. This procedure is repeated one more time so that finally there are three sudden slight drifts in the inference data which are separated by five batches.

A Dense Artificial Neural Network[5] was used to fit the artificial data because for Neural Networks, the update techniques *grad2-lstm* and *MAML* can also be applied.

The results are shown and discussed in Section III.

C. Model Adaptation Techniques used

The following model adaptation techniques are used in the comparison experiments.

- *baseline*. Refers to the original pretrained model with no update taken.
- *only-latest*. Using the model's retraining mechanism for the update and using only the data from the latest incoming data batch for the update.
- *rolling-window*. Using the model's retraining mechanism for the update and using data from the latest batch plus the data from the previous batches with a fixed sliding window size [28].
- *exp-forgetting*. Using the model's retraining mechanism for the update and using data from the latest batch plus the data from the previous batches with a fixed sliding window size. The importance of the data points decreases exponentially the older the data points are [29]. The decay factor is a hyperparameter that was optimized for the real and artificial use case separately.
- *boosting*. Boosting sequentially with the following setup: Take the original model and append a "booster model." Update only the booster model, Using a sliding window. For the real use case, in case of a Neural Network as base model, a small XGBoost model was chosen as the booster model. In case of an XGBoost model as the base model, a small XGBoost model was chosen as the booster model was chosen as well. For the artificial use case, a linear model was chosen as the booster model. The choices of the booster model types were made based on which booster model candidate yielded the better performance: Linear Regressor or XGBoost.
- *grad2-lstm*. The update technique "Learning to learn by gradient descent by gradient descent" [30]. This technique was only used with Neural Networks due to the limitations discussed section I-A (use of the gradient of

[5]Sequential, dense Neural Network with the structure: Input Layer of size 5, Dense Layer with 2 nodes and ReLu activation function, Dense Layer with 2 nodes and ReLu activation function and output layer with 1 node and ReLu activation function.

TABLE I
HYPERPARAMETERS FOR EACH UPDATE TECHNIQUE USED.

Update technique.:	Update method's hyperparameters using XGBoost	Update technique's hyperparameters using NN
only-latest-WS	learning rate	learning rate, epoch number
only-latest-SC	learning rate	-
rolling-window-WS	learning rate	learning rate, epoch number
rolling-window-SC	learning rate	learning rate, epoch number
exp-forgetting-WS	learning rate, decay factor	learning rate, epoch number, decay factor
exp-forgetting-SC	learning rate, decay factor	learning rate, epoch number, decay factor
boosting	-	learning rate when XGB as booster
grad2-lstm	-	learning rates inner optimizer and outer optimizer, epoch number
MAML	-	number of gradient steps, learning rate outer optimizer, learning rate for gradient steps
baseline	-	-

the regression model's loss function with respect to the regression model's internal parameters).

- *MAML.* This update technique uses "Model-agnostic Meta learning" [37], using Adam optimizer for outer optimization and gradient descent for inner optimization. This technique was only used with Neural Networks due to the limitations discussed in section I-A (use of the gradient of the regression model's loss function with respect to the regression model's internal parameters).

For the model's retraining mechanism in case of Neural Networks, ADAM optimizer is used [53]. For the update techniques only-latest, rolling-window and exp-forgetting, the sliding window size was chosen same as the size of the pretraining data ($165,000$ data points). For MAML and grad2-lstm, a sliding window of $5,000$ data points was used as increasing the number of update points did not lead to better accuracy. Two variants, warm-start and training from scratch, were conducted for the update techniques only-latest, rolling-window and exp-forgetting. One exception is only-latest in case of the Neural Network since it is theoretically trivial that training a Neural Network of the given size on only 30 data points will not work.

The hyperparameters of the update techniques were optimized per use case and per model. The existing hyperparameters for each technique are shown in Table I. The optimization of hyperparameters was conducted within the batch range of 0 to $3,000$ batches, as opposed to the full range up to $5,000$. This approach was adopted based on the assumption that continual hyperparameter optimization for update techniques will not be feasible in subsequent real-world applications. In the tables and figures of this work, the terms "warm-start" and "training from scratch" are abbreviated by "WS" and "SC" respectively.

III. RESULTS

The findings that were made in the course of applying the chosen model update techniques to the industrial use case are presented in the following.

TABLE II
OVERALL LOSSES FOR ALL EXPERIMENTS CONDUCTED.

Use case	Artificial case		Virtual Metrology			
Regression model	NN		NN		XGB	
Update mode	**Infm**	**Blind**	**Infm**	**Blind**	**Infm**	**Blind**
Update technique:						
only-latest-WS	0.06	0.06	0.72	0.71	2.17	3.73
only-latest-SC	-	-	-	-	2.73	3.64
rolling-window-WS	0.46	0.46	0.13	0.06	0.41	0.51
rolling-window-SC	0.49	0.47	0.39	0.33	0.34	0.40
exp-forgetting-WS	0.46	0.46	0.22	0.09	0.43	0.49
exp-forgetting-SC	0.46	0.47	0.43	0.39	0.32	0.39
boosting	0.12	0.19	0.69	0.68	0.92	0.92
grad2-lstm	0.15	0.17	0.76	0.76	-	-
MAML	0.10	0.18	0.71	0.89	-	-
baseline	0.47	0.47	0.72	0.72	0.63	0.63

TABLE III
THE VALUES SHOW HOW OFTEN THE CORRESPONDING UPDATE TECHNIQUE WAS UPDATED IN THE GIVEN USE CASE FOR THE INFORMED ADAPTATION SETTING.

Update technique:	**Artificial UC**	**Real UC NN**	**Real UC XGB**
only-latest-WS	3	24	440
only-latest-SC	-	-	739
rolling-window-WS	13	20	89
rolling-window-SC	13	15	83
exp-forgetting-WS	13	17	81
exp-forgetting-SC	13	17	83
boosting	10	20	175
grad2-lstm	13	23	-
MAML	6	26	-
baseline	0	0	0

Table II shows the results for the experiments conduced.[6] Two use cases, artificial and real (Virtual Metrology), were regarded. For both use cases, Neural Networks were used and additionally, for the real-world use case an XGBoost model was used. The loss (mean absolute error) for each subsequent batch was scaled to a reference in order to maintain confidentiality. This is defined in the following way: The normalized loss equals zero for the lowest loss of the baseline (no model updating, black curves). The normalized loss equals one for the maximum loss of the baseline. This means, the losses are shown in relative units (relative to the losses of the baseline, per use case and per model). The mean of the losses in relative units, taken over the whole inference period of $5,000$ batches, is referred to "overall loss" in this article.

In Table III it is shown for the informed case, for which model update techniques how many updates were performed during the inference (per use case and model).[7] The blind adaptaton with fixed update frequencies (updating every 5 batches for the artificial use case and every 200 batches for the real use case) led to 3 updates for the artificial and 24 updates for the real use case, for each update technique. Comparability of the values in tables II and III is given column-wise, not row-wise, due to the experiment setup.

[6]Informed is abbreviated by "Infm."
[7]Use Case is abbreviated by UC. XGBoost is abbreviated by XGB.

A. Neural Network Model for Artificial Use Case

For showing and validating the implemented update techniques on data with no unknown influences, artificial data were generated. Three times, every fifth batch, a Concept Drift was incorporated. Details about the data construction are explained in Section II-B.

1) Informed adaptation: The evolution of the loss of the Neural Network during inference for the informed update setting is shown in Fig. 1 for all update techniques applied. Model degradation, being caused by the Concept Drift, can clearly be seen in the baseline curve's increasing loss (black curve). The rolling window techniques, whether training from scratch or using warm-start, perform similar or worse than not updating at all. This is because the Concept Drifts are introduced suddenly and there is no recurrence in it by construction of the data. In the data set used for updating, the data from the newest batches are too underrepresented to make an improvement in accuracy.

Exponential forgetting warm-start (red curve) brings a slight improvement for each successive batch after a drift was introduced.

An improvement and thus successful handling of the Concept Drift is achieved by the update techniques "only-latest-WS," "grad2-lstm," "MAML" and "boosting". Using warm-start and tuning to only to the latest batch, "only-latest-WS" can adapt best to the new data. For all four update-techniques, at batch numbers 5, 10 and 15, a peak is observed in the loss curves. This is due to the experiment setup: As detector of drift, exceeding of an accuracy threshold is used. When this is detected for a batch, the update and evaluation of the updated model is performed on the next batch.

The "boosting"-technique also uses only the latest batch for updating. The booster model (model in the sequential ensemble that is closest to the output) only is updated here.

Model-Agnostic Meta Learning (MAML) gave second best results. It took two subsequent batches to update, until MAML achieved the same accuracy as "only-latest-WS." "grad2-lstm" did not achieve this accuracy, but still maintained loss values that showed clear improvement wrt. the baseline.
At this point, the correctness of the implementation especially of the techniques "MAML," "grad2-lstm" and "boosting" is verified.

2) Blind adaptation: For the case of blind adaptation, the same setting as in the last subsection is used, only that there is no trigger for the update. The numbers of the batches at which an update is performed are predefined and set to 6, 11 and 16 (for each update technique). With this setting, updating takes place one batch after the drift occurs. The resulting difference to the informed case is that after a drift occured only one update is performed for each technique, until the next drift occurs.

The results are shown in Fig. 2. It is visible that for "MAML," "grad2-lstm" and "boosting," this leads to higher losses as compared to the informed case. This means these update techniques do profit from carrying out the additional updates as in the informed case. Still, for these techniques, one update suffices to see a clear improvement as compared to the baseline of not updating at all.

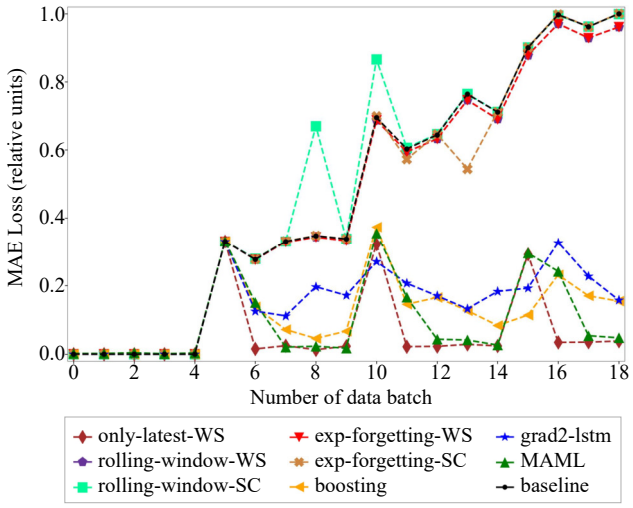

Fig. 1. Informed adaptation for the artificial use case using a Neural Network. WS= warm-start, SC= from scratch. The overall losses for each update technique are given in table II, column 2.

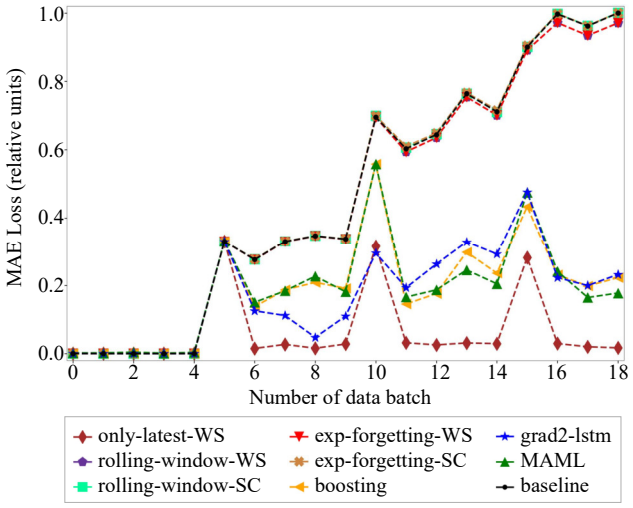

Fig. 2. Blind adaptation for the artificial use case using a Neural Network. The overall losses for each update technique are given in table II, column 3.

B. Virtual Metrology using a Neural Network

Having validated the implementation of the various update techniques and observed their behaviour on an artificial use case with known data properties and drifts, a real use case with unknown drift characteristics is regarded. The same update techniques as for the artificial use case are applied. Each update technique's hyperparameters were tuned for this use case.

1) Informed adaptation: The loss curves for the informed updating experiment for the VM case, using a Neural Network as regression model, are shown in Fig. 3. First of all, the model

degradation with time can be seen in the black curve (baseline, not updating) for which the loss increases during the inference experiment with incoming batches.

As opposed to the artificial use case, in which rolling window and exponential forgetting performed worst, in the real world use case they perform best. Especially the "rolling-window-warm-start" technique, which is the best-performing technique in this setting, is capable of restoring the initial loss.

The update technique "only-latest" which, using warm-start, fine tunes the model on the latest data batch only, results in nearly the same losses as the baseline. The "boosting" method, which also uses only the latest batch for updating, performs slightly, but not much better than the baseline (overall loss of 0.69 compared to baseline 0.72).

This shows that it is crucial for this use case to keep data points from former batches in the set of data points for updating (3300 data batches were used as sliding window). The number of batches for updating (updating-batch size) was not varied further since this is not the focus of this work.

Both techniques Model-Agnostic Meta Learning and Learning to learn by gradient descent by gradient descent, did not bring an improvement. For both cases, the optimization of the hyperparameters yielded hyperparameters that correspond to nearly not updating the model. The overall loss of grad2-lstm is slightly worse than not updating for both settings blind and informed. The loss of the MAML method is comparable to the not updating baseline for the informed case and slightly higher than the baseline for the blind case. For both, MAML and grad2-lstm, there are ranges in which they perform slightly better than the baseline. But these ranges are the exception from the overall range. The conclusion here is that for this use case, for the used setting of update frequency and batch size, the two methods do not bring improvements. In the grad2-lstm case, increasing the size of the LSTM also did not lead to any improvements.

2) Blind adaptation: In the blind adaptation setting, the rolling window warm-start technique performs best, being followed by the exponential forgetting warm-start technique (cf. Fig. 4). The difference between these two methods' overall losses is not as high as in the informed case.

The differences are likely to originate from the different update frequencies used in the blind adaptation case than those in the informed setting, resulting from excess of the accuracy threshold. Lowering the accuracy threshold for the Neural Network informed case, would presumably result in lower overall loss values. The dependency of the performance of the different update techniques on the update frequency therefore seems to be a question worth asking. For follow-up work, together with varying the update-batch size, this would be an interesting study.

The remaining update techniques qualitatively have the same relative overall losses to the exponential forgetting warm-start and rolling window warm-start techniques as well as to each other. The same conclusions that were drawn in the last subsection (informed case), hold.

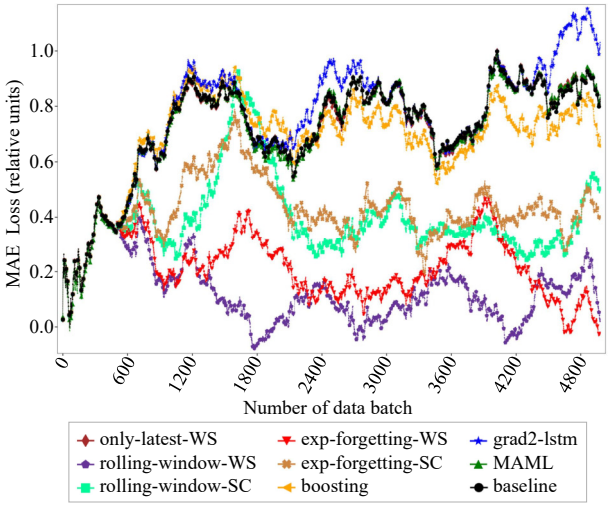

Fig. 3. Virtual Metrology regression with Neural Network – Informed adaptation setting. The overall losses for each update technique are given in table II, column 4.
To make the different lines more distinguishable, every 15^{th} data point was plotted large. However, the size of the data points has no meaning.

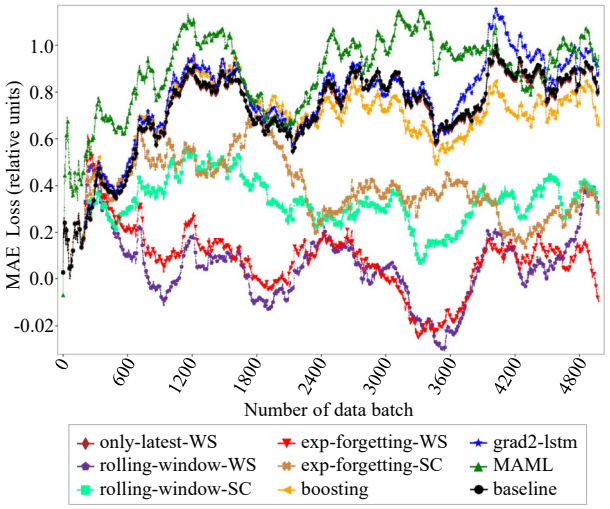

Fig. 4. Virtual Metrology regression with Neural Network – Blind adaptation setting. The overall losses for each update technique are given in table II, column 5.
To make the different lines more distinguishable, every 15^{th} data point was plotted large. However, the size of the data points has no meaning.

C. XGBoost model for Virtual Metrology

1) Informed adaptation: For the XGBoost model in the informed case (Fig. 5), the method exponential forgetting from scratch gave the best overall loss. It was followed by the method rolling window from scratch. The corresponding techniques with warm-start performed slightly worse. The "boosting" technique did not lead to improvement on the model performance with respect to the baseline but even increased the losses. Using only the data from the new batch for the model retraining ("only-latest"-techniques), made the performance even worse. This suggests that the reason for the

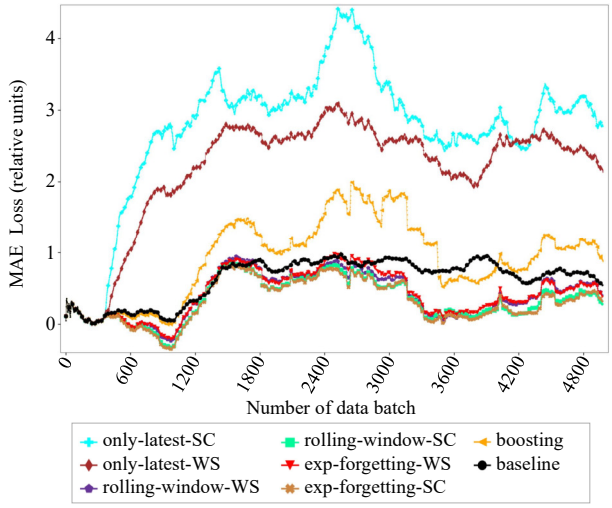

Fig. 5. Virtual Metrology regression with XGBoost model – Informed adaptation setting. The overall losses for each update technique are given in table II, column 6.
To make the different lines more distinguishable, every 30^{th} data point was plotted large. However, the size of the data points has no meaning.

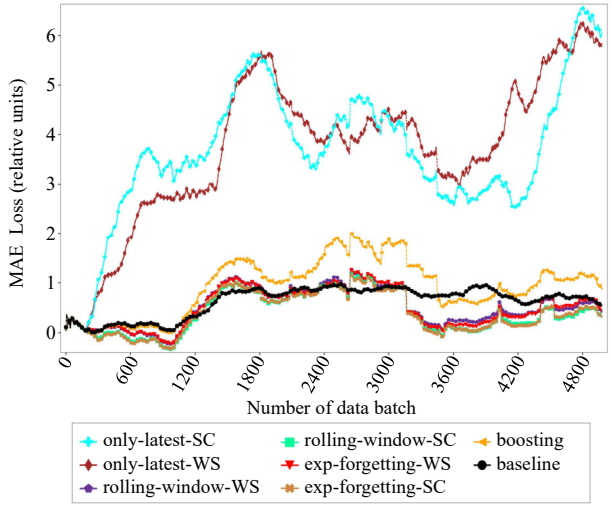

Fig. 6. Virtual Metrology regression with XGBoost model – Blind adaptation setting. The overall losses for each update technique are given in table II, column 7.
To make the different lines more distinguishable, every 30^{th} data point was plotted large. However, the size of the data points has no meaning.

low performance of "boosting" in the real use case is related to the data and not to the update strategy "boosting" in general. One batch seems not to represent the data sufficiently.

2) Blind adaptation: Similarly to the informed adaptation case, the best-performing update technique is exponential forgetting starting from scratch, being followed by rolling window starting from scratch and the two corresponding methods in warm-start setting (cf. Fig. 6). The boosting approach has the same overall accuracy as the not-updated model. Using only the latest batch for updating is not recommended as well.

IV. Summary and Conclusion

This work compares model-type-independent update techniques on regression models. For this, a regression use case from semiconductor fabrication (Virtual Metrology (VM) of a layer deposition process), was used. To validate the functionality of the update methods, a simple artificial use case (data sampled from a multilinear function) was used. On the VM use case, experiments were done with an XGBoost model and with a Neural Network. For all of these cases, a blind and an informed updating strategy was applied, yielding six scenarios that are shown in table II. Eight update techniques were regarded. All of them were successfully validated on the artificial use case. Using the Neural Network's own warm-start fine tuning method for updating, using only the latest batch, performed best in the artificial case in both, the informed and blind setting.

For the real use case (VM), using only the newest batch for updating performed poorly in all scenarios. For both, the XGBoost and Sequential Neural Network, sliding window techniques performed best in the VM case. In case of the XGBoost model, it did not make a significant difference if training from scratch or warm-start was used. For the Neural Network, warm-start clearly performed better than training from scratch. This is explainable by the good transfer learning capability of Neural Networks ([14], p.12). The boosting approaches lead to slight improvement for the Neural Network and to no improvement for the XGBoost model.

The best technique for VM, XGBoost, was exponential forgetting, training from scratch. For the Neural Network in the VM case, rolling window with warm-start performed best.

In this work, the batch size used for updating and the updating frequency were set to constant values. Nevertheless, comparing the performances of informed and blind scenarios for the real use case leads to the (obvious) conclusion that which update method performs best, depends on the frequency of the updates as well as the batch size of the updates. A study varying these hyperparameters would be an interesting follow-up work.

For the Neural Network, two Meta-learning update techniques were used in this work: "Model-agnostic meta learning" and "Learning to learn by Gradient descent" [37], [30]. Both of them were not able to improve the loss as compared to the base line. It is important to note that their implementation was validated on the artificial use case, on which they showed significant improvement. Chen et. al. conducted an experiment in which they trained a) a Multilayer Perceptron and b) a Convolutional Neural Network on the MNIST dataset, using different optimizers [44], [35]. Among these optimizers, the approach "Learning to Learn ba Gradient Descent" from Andrychowicz, called grad2-lstm in this article, was used as well. In the experiment of [44], on the Multilayer Perceptron, the optimizer did not converge. This result is similar to the result that was obtained in the article at hand for the grad2-lstm method, where it did not provide loss improvement in the real world use case. The experiment from [44] worked for

the Convolutional Neural Network they used, but only for a small number of iterations. It is important to emphasize that, in the work at hand, no Convolutional Network is used and the task that the optimizee performs is not classification, but regression.

Although the Meta-Learning techniques used, MAML and grad2-lstm, did not show improvement for this real use case and setting (meaning fixed updating batch size and updating frequency), it might be good to not discard them when doing studies with varying batch size for updating. This is because they might outperform other techniques for smaller updating batch sizes (in the case of few-shot learning).

The insight gained from the experiments conducted is there is no "one update technique" which performs best in all experiments. For the Virtual Metrology use case, the group of sliding window techniques has superior performance. For the artificial use case, this group performed worse than other techniques. For the artificial use case, the bad performance of the sliding window techniques is explainable by the drift properties: The Concept Drift incorporated into the artificial use case consists of slight sudden jumps in the function generating the training data. Such a behaviour can be theoretically expected in real use cases as well, due to maintenance events. In practice, not only maintenance events are the cause of drifts, but a variety of other influences might change the response surface as well over time. To explain why sliding window techniques performed well on the real use case, and others performed worse (for instance MAML and grad2-lstm), the properties of the real use case's data and their drift properties could be beneficial. The examination of the use case's data and drift properties were not the focus of this work though. This work aims at finding the most suitable update technique for the given use case for different possible models. This is because scalability to other use cases, for which different model types might be best-suited, is desirable. The focus therefore lied on examining the performance (accuracy) of model-independent update techniques. More knowledge about scalability of update methods to other use cases and models can be gained by performing comparable studies on other real world use cases, for example other Virtual Metrology cases. It would be possible to examine, if for each use case the same update techniques perform best, and if not, to ask what causes the differences. Presumably, it might be helpful to examine data and drift properties of the different use cases in course of this, which makes this topic an interesting task for future work. With a wider range of experiments (on more use cases, with more model types and with variation of update batch size and update frequency), recommendations for new use cases (potentially using different model types) could be given and thus scaling of the update technique application can be aided. The work at hand provides a starting point for this research.

ACKNOWLEDGMENT

This work is funded by the European Union within "NextGeneration EU", by the Federal Ministry for Economic Affairs and Climate Action (BMWK) on the basis of a decision by the German Bundestag and by the State of Saxony with tax revenues based on the budget approved by the members of the Saxon State Parliament in the framework of "Important Project of Common European Interest - Microelectronics and Communication Technologies", under the project name "EUROFOUNDRY".

REFERENCES

[1] V. Maitra, Y. Su, and J. Shi, "Virtual metrology in semiconductor manufacturing: Current status and future prospects," *Expert Systems with Applications*, vol. 249, p. 123559, 2024. doi: 10.1016/j.eswa.2024.123559

[2] S. Yan, C. Luo, S. Wang, S. Ding, L. Li, J. Ai, Q. Sheng, Q. Xia, Z. Li, Q. Chen, S. Li, H. Dai, and Y. Zhong, "Virtual metrology modeling for cvd film thickness with lasso-gaussian process regression," in *2023 China Semiconductor Technology International Conference (CSTIC)*, 2023. doi: 10.1109/CSTIC58779.2023.10219236 pp. 1–4.

[3] C. Schröer, F. Kruse, and J. M. Gómez, "A systematic literature review on applying crisp-dm process model," *Procedia Computer Science*, vol. 181, pp. 526–534, 2021. doi: 10.1016/j.procs.2021.01.199

[4] F. Bayram, B. S. Ahmed, and A. Kassler, "From concept drift to model degradation: An overview on performance-aware drift detectors," *Knowledge-Based Systems*, vol. 245, p. 1, 2022. doi: 10.1016/j.knosys.2022.108632

[5] A. L. Suárez-Cetrulo, D. Quintana, and A. Cervantes, "A survey on machine learning for recurring concept drifting data streams," *Expert Systems with Applications*, vol. 213, p. 118934, 2023. doi: 10.1016/j.eswa.2022.118934. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422019522

[6] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, pp. 1–37, Mar. 2014. doi: 10.1145/2523813

[7] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2019. doi: 10.1109/TKDE.2018.2876857

[8] A. Choudhary, P. Jha, A. Tiwari, and N. Bharill, "A brief survey on concept drifted data stream regression," in *Soft Computing for Problem Solving*, A. Tiwari, K. Ahuja, A. Yadav, J. C. Bansal, K. Deep, and A. K. Nagar, Eds. Singapore: Springer Singapore, 2021. doi: 10.1007/978-981-16-2712-5_57 pp. 733–744.

[9] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018. doi: 10.1109/MSP.2017.2765202

[10] Y. Song, G. Zhang, J. Lu, and H. Lu, "A fuzzy kernel c-means clustering model for handling concept drift in regression," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017. doi: 10.1109/FUZZ-IEEE.2017.8015515 pp. 1–6.

[11] D. Liu, Y. Wu, and H. Jiang, "Fp-elm: An online sequential learning algorithm for dealing with concept drift," *Neurocomputing*, vol. 207, pp. 322–334, 2016. doi: 10.1016/j.neucom.2016.04.043

[12] S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle, "A case-based technique for tracking concept drift in spam filtering," in *Applications and Innovations in Intelligent Systems XII*, A. Macintosh, R. Ellis, and T. Allen, Eds. London: Springer London, 2005. doi: 10.1007/1-84628-103-2_1. ISBN 978-1-84628-103-7 pp. 3–16.

[13] Y. Song, G. Zhang, H. Lu, and J. Lu, "A noise-tolerant fuzzy c-means based drift adaptation method for data stream regression," in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2019. doi: 10.1109/FUZZ-IEEE.2019.8859005 pp. 1–6.

[14] J. Vanschoren, "Meta-learning: A survey," *arXiv preprint arXiv:1810.03548*, 2018. doi: 10.48550/arXiv.1810.03548

[15] J. Son, S. Lee, and G. Kim, "When meta-learning meets online and continual learning: A survey," 2023. doi: 10.48550/arXiv.2311.05241

[16] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," 2019.

[17] S. Lee, H. Jeon, J. Son, and G. Kim, "Sequential bayesian continual learning with meta-learned neural networks," 2024. [Online]. Available: https://openreview.net/forum?id=6r0BOIb771

[18] J. von Oswald, C. Henning, B. F. Grewe, and J. Sacramento, "Continual learning with hypernetworks," 2022. doi: 10.48550/arXiv.1906.00695

[19] K. Li and J. Malik, "Learning to optimize," 2016. doi: 10.48550/arXiv.1606.01885

[20] H. M. Gomes, J. P. Barddal, L. E. B. Ferreira, and A. Bifet, "Adaptive random forests for data stream regression." in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2018. [Online]. Available: https://www.ppgia.pucpr.br/~jean.barddal/assets/pdf/arf_regression.pdf

[21] J. Montiel, R. Mitchell, E. Frank, B. Pfahringer, T. Abdessalem, and A. Bifet, "Adaptive xgboost for evolving data streams," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020. doi: 10.1109/IJCNN48605.2020.9207555 pp. 1–8.

[22] F. M. de Souza, J. Grando, and F. Baldo, "Adaptive fast xgboost for regression," in *Intelligent Systems*, J. C. Xavier-Junior and R. A. Rios, Eds. Cham: Springer International Publishing, 2022. doi: 10.1007/978-3-031-21686-2_7. ISBN 978-3-031-21686-2 pp. 92–106.

[23] J. Zheng, F. Shen, H. Fan, and J. Zhao, "An online incremental learning support vector machine for large-scale data," *Neural Computing and Applications*, vol. 22, pp. 1023–1035, 2013. doi: 10.1007/s00521-011-0793-1

[24] Łukasz Korycki and B. Krawczyk, "Adaptive deep forest for online learning from drifting data streams," 2020. doi: 10.48550/arXiv.2010.07340

[25] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "One-shot learning with memory-augmented neural networks," 2016. doi: 10.48550/arXiv.1605.06065

[26] S. Xu and J. Wang, "Dynamic extreme learning machine for data stream classification," *Neurocomputing*, vol. 238, pp. 433–449, 2017. doi: 10.1016/j.neucom.2016.12.078

[27] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," 2018. doi: 10.48550/arXiv.1707.03141

[28] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '02. New York, NY, USA: Association for Computing Machinery, 2002. doi: 10.1145/543613.543615. ISBN 1581135076 p. 1–16.

[29] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weighting," *Intell. Data Anal.*, vol. 8, pp. 281–300, 2004. doi: 10.3233/IDA-2004-8305

[30] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/fb87582825f9d28a8d42c5e5e5e8b23d-Paper.pdf

[31] N. Oza, "Online bagging and boosting," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, 2005. doi: 10.1109/ICSMC.2005.1571498 pp. 2340–2345 Vol. 3.

[32] E. Lughofer, "Efficient sample selection in data stream regression employing evolving generalized fuzzy models," in *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2015. doi: 10.1109/FUZZ-IEEE.2015.7337844 pp. 1–9.

[33] Y. Song, G. Zhang, J. Lu, and H. Lu, "A fuzzy kernel c-means clustering model for handling concept drift in regression," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, 2017. doi: 10.1109/FUZZ-IEEE.2017.8015515 pp. 1–6.

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, nov 1997. doi: 10.1162/neco.1997.9.8.1735

[35] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012. doi: 10.1109/MSP.2012.2211477

[36] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf

[37] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1126–1135. [Online]. Available: https://proceedings.mlr.press/v70/finn17a.html

[38] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. de Freitas, "Learning to learn without gradient descent by gradient descent," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017. doi: 10.5555/3305381.3305459 p. 748–756.

[39] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01. New York, NY, USA: Association for Computing Machinery, 2001. doi: 10.1145/502512.502568. ISBN 158113391X p. 377–382.

[40] J. Z. Kolter and M. A. Maloof, "Dynamic weighted majority: An ensemble method for drifting concepts," *The Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007. [Online]. Available: http://jmlr.org/papers/v8/kolter07a.html

[41] M. P. S. Bhatia, "A two ensemble system to handle concept drifting data streams: recurring dynamic weighted majority," *International Journal of Machine Learning and Cybernetics*, vol. 10, 03 2019. doi: 10.1007/s13042-017-0738-9

[42] A. Liu, J. Lu, and G. Zhang, "Diverse instance-weighting ensemble based on region drift disagreement for concept drift adaptation," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 293–307, 2020. doi: 10.1109/TNNLS.2020.2978523

[43] B. Celik and J. Vanschoren, "Adaptation strategies for automated machine learning on evolving data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, p. 3067–3078, Sep. 2021. doi: 10.1109/tpami.2021.3062900

[44] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin, "Learning to optimize: A primer and a benchmark," *Journal of Machine Learning Research*, vol. 23, no. 189, pp. 1–59, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-0308.html

[45] S. Wang, J. Sun, and Z. Xu, "Hyperadam: A learnable task-adaptive adam for network training," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 5297–5304, 07 2019. doi: 10.1609/aaai.v33i01.33015297

[46] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016. doi: 10.48550/arXiv.1609.04747

[47] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Networks for Machine Learning*, vol. 4, pp. 26–31, 2012. [Online]. Available: https://cir.nii.ac.jp/crid/1370017282431050757

[48] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017. doi: 10.5555/3305890.3306069 p. 3751–3760.

[49] K. Lv, S. Jiang, and J. Li, "Learning gradient descent: better generalization and longer horizons," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017. doi: 10.5555/3305890.3305913 p. 2247–2255.

[50] T. Chen, W. Zhang, Z. Jingyang, S. Chang, S. Liu, L. Amini, and Z. Wang, "Training stronger baselines for learning to optimize," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020. doi: 10.5555/3495724.3496339 pp. 7332–7343.

[51] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016. doi: 10.1145/2939672.2939785. ISBN 978-1-4503-4232-2 pp. 785–794.

[52] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, doi: 10.48550/arXiv.1603.04467, Software available from tensorflow.org.

[53] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014. doi: 10.48550/arXiv.1412.6980