DNN Classifiers Adaptation to Pattern of Life Tasks using Instances and Elastic Weight Consolidation

Abanoub Ghobrial
Trustworthy Systems Laboratory
University of Bristol
Dept. of Computer Science, Bristol, UK
abanoub.ghobrial@bristol.ac.uk

Xuan Zheng
Trustworthy Systems Laboratory
University of Bristol
Dept. of Computer Science, Bristol, UK
dq18619@bristol.ac.uk

Hamid Asgari
Technology and Innovation Research,
Thales
Reading, UK
hamid.asgari@uk.thalesgroup.com

Kerstin Eder
Trustworthy Systems Laboratory
University of Bristol
Dept. of Computer Science, Bristol, UK
kerstin.eder@bristol.ac.uk

***Abstract*—Autonomous systems (AS) use Deep neural network (DNN) classifiers to allow them to operate in complex, high dimensional, non-linear, and dynamically changing environments. Due to the complexity of these environments, DNN classifiers may often output misclassifications due to experiencing new tasks in their operational environments, which were not identified during development. Removing a system from operation and retraining it to include the new identified task is an expensive process and becomes uneconomically feasible as the number of such autonomous systems increase. Additionally, such misclassifications may cause accidents causing loss of lives or massive financial losses.**

In this paper, we propose to tackle this issue by investigating if such DNN classifiers can adapt its knowledge to fit the autonomous system's pattern of life (PoL) using limited number of instances encountered during operation. This hence increases the systems reliability (and thus trustworthiness) in its operational environment. Retraining DNNs on different observations than used in prior training is known to cause catastrophic forgetting or significant model drift. We investigate if this problem can be controlled by using elastic weight consolidation (EWC) whilst learning from new limited samples. Results show that using EWC does make the process of PoL adaptation a lot more controlled, and thus allowing for more reliable PoL adaption of autonomous systems.

***Index Terms*—Adaptive, Classifiers, Autonomous systems, Pattern of life, Elastic weight consolidation.**

I. Introduction

Autonomous systems (AS) are being developed using deep neural network (DNN) classifiers to interact and adapt in dynamically changing real-world environments to achieve their intended goals. The benefit of DNNs driving their usage in autonomous systems is their ability to learn complicated patterns in high-dimensional data from complex environments, and thus produce highly non-linear decision boundaries to cope with the complexity of operational environments. However, this also causes the trustworthiness of such systems to be low, as they are very difficult to verify. Generally, trustworthiness increases with increase in reliability[1], which is often demonstrated through established verification and validation procedures. A popular example of such ASs, are self-driving cars. Current research shows that for each self-driving car an economically impossible amount of testing is required to verify the system for deployment [1]. For which innovative methods of increasing the efficiency of testing and validation are actively being developed to make the process more economically viable e.g. [2, 3].

On the other hand, the work presented herein aims at increasing the reliability, and hence trustworthiness, of an AS on its operational environment by increasing the performance on its *pattern of life* (PoL) tasks. PoL is a term used widely in the field of intelligence generation, to refer to intelligence generated by observing and analysing behaviours of any system over a period of time [4]. We use the term PoL tasks to refer to tasks that an AS encounters during operation. Specifically, we are interested in PoL tasks that come from a slightly different distribution than used in initially training the DNN classifiers and thus causes misclassifications. The method of adapting to PoL tasks complements the verification and validation work on autonomous systems to achieve deployment. This follows an approach presented by Koopman et.al. [5] , which says given that an autonomous system passes some minimum safety validation case, the system is deployed and is improved during operation to increase its reliability over time.

Upon a DNN classifier retraining to learn new information during operation, a reassessment of the safety validation case and testing for verification would be required, which can be impractical to do during operation after every retrain. How to show the safety compliance of an evolving DNN during operation against a set of requirements or regulations is beyond the scope of this paper, but can be achieved through the use of runtime safety behavioural checkers as was presented by Harper et.al. [6]. Also, it can be predicted that a system may exhibit undesirable or unsafe behaviours by assessing how dissimilar the operational input data to the data used at training or retraining of DNN classifiers as was demonstrated

[1]https://www.vocabulary.com/dictionary/trustworthiness

by Hond et.al. [7] and other similar works e.g. [8, 9, 10].

DNN classifiers training uses gradient based optimisation algorithms to learn, which hinders the process of developing reliable adaptive autonomous systems using only encountered observations. These are discussed in the two bullet points below:

- The gradient optimiser will modify the decision boundary based on the choice of observations used in training or retraining. Thus as the observations used in retraining the classifier change between different operational environments, the decision boundary will adapt accordingly, which may result in the reliability of the DNN classifier changing as the system alternates between operational environments. This can be tackled through the usage of PoL adapted classifiers, motivated by our work, tagged to each operational environment of the system and decoupled from each other.
- When a DNN classifier encounters a situation, which results in a misclassification, there is a few observations that can be as little as a single observation to use for retraining (adaptation). Retraining using the available new observations results in significant model drift or in a phenomenon known as *catastrophic forgetting* [11]. To generally avoid that, the new observations are added to the original dataset used in initial training and the classifier is fully retrained. Full retraining, however, can be very cumbersome to do during operation. Alternatively, the *learning rate*[2] can be set to be very low in order to avoid the model from drifting. This unfortunately causes the DNN model not ti learn information sufficiently from new observations because the learning rate is low, restricting the model from drifting to adapt to new information. Trying to optimise the learning rate still results in significant if not catastrophic forgetting, as shown later by the results. This raises the need for a method to protect features of the DNN classifier important for preserving performance on old information, whilst still providing flexibility for the model to drift and adapt to PoL tasks. There are several methods in the literature that have been demonstrated to be able to protect previous information in DNNs,.e.g. elastic weight consolidation (EWC) [12] is one of the very successful methods at doing so and is the method we have adopted in this paper. Further information on the rationale behind choosing EWC is provided in later sections of the paper.

This paper aims at discussing the following two research questions:

Research question 1: Can the use of EWC allow for higher learning rates to be used during retraining to allow for both i) the DNN classifier to improve its benefit from learning new encountered observations and ii) whilst controlling the forgetting or model drift on previous knowledge?

[2]The hyper-parameter used in controlling how much the neural network model is flexible to drift. The lower the learning rate the less drift is allowable.

Research question 2: Based on a positive outcome of research question 1, can this be used to allow for pattern of life adaptation of autonomous systems?

The process of continual learning or adaptation of an autonomous system also involves the detection of incorrect behaviours and supply of correct labels for the new observations encountered and used in retraining. This can be achieved through the use of oracles [13, 14] to provide a ground truth reference and we aim to investigate this further in upcoming works, but in this paper we assume that the correct labels are available for misclassified observations.

A. Related work

The problem of forgetting is a well-known issue with DNNs, which prevents the application of continual learning in autonomous systems. Over the past decade, there has been an increase in focus by researchers on the topic of forgetting in DNNs. In continual learning, full retraining is usually resorted to as it gives the most guarantee that no previous information will be lost, where new observations are simply added to the original dataset and the neural network is trained from scratch e.g. [15]. Whilst this might be the easiest approach, it is limited to certain types of systems and is not sustainable for dynamic autonomous systems where training on the original tasks takes hours or days to complete. The increase in interest on the topic of forgetting has also been encouraged by published research investigating the limitations of continual learning in gradient optimised DNNs, e.g. Goodfellow et.al. [11], where they studied the effect of the relationship between different learnt tasks and catastrophic forgetting for different optimisation algorithms in the literature.

In order to solve these limitations of continual learning, researchers have explored numerous techniques that try to conserve or protect previously learnt information when learning incrementally instead of full retraining. Conserving learnt information can be achieved by freezing weights e.g. [16, 17], or introducing penalisation terms that restrain weights of importance to old information in the DNN from significantly changing. Thus optimising the performance of a DNN between both the previous and new learnt information e.g. [12, 18, 19].

Inspired from Dopaminergic neurons in mammalian brains which allows for rapid and isolated adaptation in the synapses of neurons, Allred et.al. [20] tried to solve the problem of catastrophic interference in Spiking Neural Networks (SNNs). They introduced an artificial dopaminergic neuron in each layer of the spiking neural networks. The role of which is to control neurons that are allowed to adapt during retraining. If a neuron output is above a certain threshold for previously learnt information, then it is deemed as important for old tasks and should not be altered, thus the artificial dopaminergic neuron stops it from changing during retraining on new information, whilst allowing neurons of less or no importance to previous tasks to be altered. By measuring the firing threshold in neurons they also detect when novel data is experienced.

Elastic weight consolidation (EWC), [12], is another approach for continual learning developed for DNNs that has

shown very promising results in the literature. EWC uses a Bayesian framework to learn continuously, whereby the old information learnt is approximated in a term and included in the loss function used in retraining. Such that neurons that have more significance on the decision-making of old information have less flexibility to deviate. Therefore, as new information is learnt the EWC framework allows the previously trained model to drift in a direction that allows it to optimise for both old and new information simultaneously.

These approaches discussed above are examples that fall under the category of regulatory techniques in continual learning, mainly focusing on modifying existing neural network resources to learn new information whilst minimising forgetting. A comprehensive and thorough review is provided by Parisi et.al. [21], where they cover other continual learning categories such as architectural techniques, whereby existing neurons trained on previous information freeze and new neurons are added to learn new information. They also cover rehearsal and memory replay approaches which cover methods that try to mitigate the problem of forgetting by retraining the system occasionally on subsets of previously learnt observation to avoid the model from completely drifting from previously learnt information.

Kemker et.al. [22] introduced their own metrics to compare between different developed schemes that avoid catastrophic forgetting. Based on their experimentation they reach the conclusion that the achieved optimal performance for each of the developed schemes varies depending on the incremental training paradigm and type of data used. They also conclude based on their results that all the techniques investigated demonstrate that the challenge of catastrophic forgetting is not yet solved. This demonstrates that there is no one-size-fits-all solution that can generalise to all dynamically changing environments of an autonomous system. Therefore a solution ought to be developed specifically for each of the changing environments of an autonomous system, which is similar to how the human brain adapts to learn thoroughly tasks and rare instances in its operational environments or pattern of life.

B. Contribution and structure of the paper

In this paper, we develop a retraining pipeline for PoL adapting DNN classifiers. A shortcoming of many of the experimentation in the literature is that it is always assumed that sufficient data is available for learning the new information. This is not true with real time systems that may have as little as one observation to retrain on. We do not make this assumption in our experimentation and retrain on limited instances of new information.

We control the forgetting of older information by using EWC which is shown to be one of the best optimising techniques for overcoming catastrophic forgetting based on Parisi et.al. [21]. The retraining pipeline learns in a fashion that tries to balance between the original tasks and the PoL tasks encountered during operation based on some chosen criteria.

In the next section, we introduce some preliminaries that will help in understanding the rationale behind the way the retraining pipeline was created. Section III introduces the methodology, which includes the challenges, the pipeline, and the rationale behind it. Experimentation is presented in Section IV, along with the results and discussion. We conclude and discuss future works in sections V and VI, respectively.

II. Preliminaries

To gain an understanding of the requirements and limitations of SGD and EWC we go through their derivations in this section and note down any assumptions that the SGD or EWC were based on, to ensure they are satisfied in the developed methodology. Also, some definitions are outlined that will be used in subsequent sections.

A. Definitions

1) Training and testing datasets: The set of observations used in training the DNN are referred to as the training dataset. These observations are random samples from the distribution representing the operational environment of the system. Equivalently, a test dataset refers to other random samples from the same distribution used to validate how well the trained network has generalised to the whole distribution. Training and testing datasets have *labels* associated with them, which provides a reference to the correct classification during constructing or validating the DNN.

2) Classes: Represents the possible output classifications of a DNN, which form architecturally the last layer in a DNN. A new class represents a completely *new classification* that falls out of the set of classifications of the DNN. Therefore, if a new class is to be added to a DNN, a change in the architecture will be required at least for the last layer of the DNN.

3) A Task: In the continual learning literature, the word *task* is used very subjectively, depending on the context of the paper. We differentiate between the different definitions of tasks in the literature as follows: a *novel pattern task* to refer to different distributions falling under the same classes (i.e. same labels) of a trained DNN but follows a completely different pattern. For example, the MNIST [23], Permuted-MNIST [11] and Fashion-MNIST [24] datasets can be thought of as distributions having the same labels but different patterns. On the other hand we use the term *novel noise task* to refer to distributions following a similar pattern to the trained data but with some noise applied to it e.g. the n-MNIST dataset [25].

We use the default terms *task* and *new task* to refer to tasks with similar patterns but different noises, unless stated otherwise. e.g. digit 9 with blur noise from the n-MNIST would count as a new task compared to the digit 9 from the MNIST dataset.

4) An Instance: Is a single or many oversampled observations from an existing task or a new task. A *new instance* means that the observations are misclassified by the DNN.

B. Stochastic Gradient Descent (SGD)

Figure 1 shows the computation in a single artificial neuron of a DNN. Where $[x_1, ..., x_K]$ and $[w_1, ..., w_K]$ are the set of inputs and weights into the neuron respectively. The term b

is the bias and x_{out} is the output of the neuron as shown by Equation 1. ReLU is the popular rectified linear unit activation function used in DNNs given by Equation 2. Refer to [26] for a more thorough formal representation of DNNs with several neurons and layers.

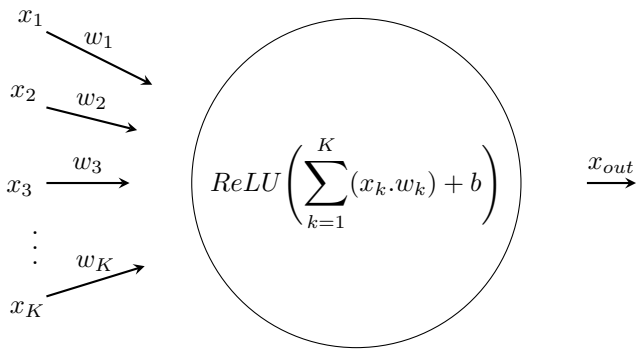


Fig. 1: Artificial neuron in a DNN

$$x_{out} = ReLU\left(\sum_{k=0}^{K}(x_k.w_k)+b\right) \tag{1}$$

$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Training a DNN refers to tweaking the values of the weights and the biases for each neuron in a DNN so that it achieves the desired outputs for the provided observations (training dataset). This is achieved by minimising a *loss function*, $\mathcal{L}$, that compares between the DNNs predicted output $\hat{y}$ (for a neural network comprising of one neuron $\hat{y} = x_{out}$) and the target output y i.e. the label. A method for doing this comparison is by using the mean squared error given by Equation 3, where N is the number of observations used in training.

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}(y_i - \hat{y}_i)^2 \tag{3}$$

The aim during training is to minimise the difference between the predicted outputs of the DNN and the target output, which means to identify where a local minimum is i.e. gradient is zero. This is achieved numerically using gradient-based optimisation [27] methods.

Substituting the predicted output calculation in place of $\hat{y}$ and using the chain rule it can be shown that the gradient-based optimisation used in training most DNN classifiers is given by Equation 4 (Refer to [28] for the full derivation). Where θ represents the set of weights and biases in a DNN, η is a hyperparameter known as the *learning rate* and t is the training step ID taken in optimising the DNN.

$$\theta_{t+1} = \theta_t - \eta\frac{\partial\mathcal{L}(\theta)}{\partial\theta} \tag{4}$$

The stochastic gradient descent (SGD) optimiser algorithm implements Equation 4, but selects random (stochastic) mini batches from the observations and optimise using gradient descent. This is instead of optimising on all the available N observations, hence increasing the computation efficiency of training. Practically when training a DNN, the samples provided to the SGD algorithm have to be random from the overall distribution. This allows for the gradient to move in a direction that optimises the DNN for the whole distribution rather than for a subset of it.

Assumption 1: The number of observations N available during any training session is sufficient to represent the overall required distribution of the operational environment, thus allowing for the SGD optimiser to optimise to a sufficient local minimum.

C. Elastic Weight Consolidation (EWC)

Elastic weigh consolidation presented first by Kirkpatrick et.al. [12] is an algorithm that aims at protecting weights important for correct classification of previously learnt data distributions.

1) Derivation: During training of a DNN the goal is to minimise the loss function $\mathcal{L}(\theta)$, represented as the Log-Likelihood function $-log(P(\theta|D))$ [27]. This aims at estimating θ, which is the set of weights and biases in a DNN, given D, the dataset representing the samples of the distribution of interest. D can be split into two independent datasets such that $D = \{D_A, D_B\}$, where D_A and D_B are datasets that are trained on sequentially and each of them may represent a different distribution. Using the chain rule in probability it can be shown that:

$$\begin{aligned} log(P(\theta|D_A, D_B)) = log(P(D_B|\theta, D_A)) + log(P(\theta|D_A)) \\ - log(P(D_B|D_A)) \end{aligned} \tag{5}$$

Considering the RHS of equation 5:

- First term, using conditional independence $log(P(D_B|\theta, D_A)) = log(P(D_B|\theta))$ and hence can be seen as the loss function, $\mathcal{L}_{\mathcal{B}}(\theta)$ that needs to be minimised for the new distribution or dataset D_B alone.
- Second term, $-log(P(\theta|D_A))$ is the loss function for training the neural network on distribution D_A only. Thus can be denoted as $\mathcal{L}_{\mathcal{A}}(\theta)$.
- Third term, is irrelevant as this term is constant with respect to θ and thus is lost when optimising using the stochastic gradient descent (SGD) i.e. does not need to be computed. We will neglect this term for the rest of the derivation.

Therefore, the overall loss function in equation 5 can be written as:

$$\mathcal{L}(\theta) = \mathcal{L}_{\mathcal{B}}(\theta) + \mathcal{L}_{\mathcal{A}}(\theta) \tag{6}$$

In continual learning, distribution A would have been trained on initially, and later samples from distribution B arises and is required to be learnt by the DNN. In this case, the

term $\mathcal{L}_{\mathcal{A}}(\theta)$ is considered to be intractable as it is assumed that access to training samples for distribution A is not available after initial training.

The underlying idea of EWC is to take a Bayesian approach to adapt the DNN model parameters, therefore learning additional distributions whilst avoiding catastrophic forgetting or minimising forgetting. However, due to intractable terms, it is not possible to maintain the full posterior $P(\theta|D))$. An inference technique is required to approximate these intractable terms.

EWC can be seen as an on-line approximate inference algorithm [29]. An essential assumption for EWC to approximate $\mathcal{L}_{\mathcal{A}}(\theta)$ is that the DNN has been optimised for D_A such that θ has reached a local or a global minimum, θ^*_A, for distribution D_A. This allows for $-log(P(\theta|D_A))$ to be approximated as a Gaussian distribution function at its mode using Laplace's method [30]. Expanding $-log(P(\theta|D_A))$ using Taylor series around θ^*_A:

$$\begin{aligned} -log(P(\theta|D_A)) \approx & - log(P(\theta^*_A|D_A)) \\ & + (\frac{\partial(-log(P(\theta|D_A)}{\partial\theta}|_{\theta^*_A})(\theta - \theta^*_A) \\ & + \frac{1}{2}(\theta - \theta^*_A)^T H(\theta^*_A)(\theta - \theta^*_A) \\ & + \cdots \end{aligned} \tag{7}$$

Considering the RHS of equation 7:

- First term, is a constant and similar to earlier it will get lost in the SGD optimiser.
- Second term, evaluates to gradient 0 as it is assumed that θ^*_A is at the mode of the distribution.
- Third term; $H(\theta^*_A)$ is the Hessian of $-log(P(\theta|D_A))$ with respect to θ evaluated at θ^*_A, which is $(\frac{\partial^2(-log(P(\theta|D_A)}{\partial\theta}|_{\theta^*_A})$.

The Hessian can be computed by approximating it to the empirical Fisher information matrix. Using Bayesian rule:

$$\begin{aligned} H(\theta^*_A) = & - \frac{\partial^2(log(P(D_A|\theta)))}{\partial\theta^2}\bigg|_{\theta^*_A} \\ & - \frac{\partial^2(log(P(\theta)))}{\partial\theta^2}\bigg|_{\theta^*_A} \\ & + \frac{\partial^2(log(P(D_A)))}{\partial\theta^2}\bigg|_{\theta^*_A} \end{aligned} \tag{8}$$

Considering the RHS of equation 8:

- First term can be approximated as the negative of the empirical Fisher information matrix, F, [31, 32, 33]. The Fisher can be defined as a way of measuring the amount of information that an observable random sample $D_A[n]$ carries about a set of unknown parameters θ of a distribution that models $log(P(D_A|\theta)$, where n is an index falling within the size, N, of the observable random samples D_A. Formally, it is the negative of the expected value of the observed information, hence it can be shown that it approximates to the first term of equation 8:

$$\begin{aligned} F(\theta) &= -NE\left[\frac{\partial^2(log(P(D_A[n]|\theta)))}{\partial\theta^2}\right] \\ &\approx -N\frac{1}{N}\sum_{n=1}^{N}\frac{\partial^2(log(P(D_A[n]|\theta)))}{\partial\theta^2} \\ &= -\sum_{n=1}^{N}\frac{\partial^2(log(P(D_A[n]|\theta)))}{\partial\theta^2} \\ &= -\frac{\partial^2(log(P(D_A|\theta)))}{\partial\theta^2} \end{aligned} \tag{9}$$

  The approximation made to the expectation in equation 9 becomes exact as the number of observations or samples becomes infinite. Therefore, the data size N of the previous information is crucial to the applicability of using the EWC approximation.
- Second term, is the *prior probability*.That is the probability distribution the DNN represents before being trained on any observations i.e. datasets. Given that often θ in DNNs are initialised using a random uniform distribution, then this term evaluates to zero, and hence is ignored by the EWC algorithm.
- Third term, evaluates to zero as non-dependent on θ.

Putting terms together from the previous steps makes equation 6 reach the EWC loss function presented by [12]:

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_j \frac{\lambda}{2}F^A_j(\theta_j - \theta^*_{A,j})^2 \tag{10}$$

where λ is a hyper-parameter presented by Kirkpatrick et.al. to allow for fine tuning to minimise forgetting. We summarise the list of assumptions for which equation 10 holds:

Assumption 2: The DNN was trained very well on the previous distribution represented by D_A that θ has reached a local or a global minimum i.e. $\theta^*_A = argmin_\theta\{-log(P(\theta|D_A))\}$.

Assumption 3: "Enough" observations are available in D_A to allow for the approximation from the Hessian to the empirical Fisher information matrix.

2) The choice of EWC: Going through the derivation above, which adopts a Bayesian frame work, tries to approximate the previously learnt data and incorporate it into the loss function to allow the optimiser to vary the DNN parameters θ in a way that *balances* between previously learnt distributions and updated distributions experienced by new observations. This balance is controlled via the hyper-parameter λ.

Adopting a Bayesian framework, which in essence gives a model the ability to increase or decrease its confidence on existing knowledge and update this knowledge based on new observations, makes EWC very suitable for the application of PoL adaptation. It also ticks the boxes of being memory and training efficient compared to other retraining schemes that were evaluated by Kemker et.al. [22]

III. Methodology

The three assumptions outlined in the previous section forms the grounds for the rationale of the developed retraining pipeline shown in this section.

A. The challenge with continual learning for PoL adaptation

Each task that needs to be learnt can be viewed as a distribution for which observations exist in the training dataset. For a DNN to learn this task or distribution, it optimises its weights and biases by minimising the loss function as was explained earlier in section II-B, to represent this distribution. When having several tasks, that means several distributions are required to be learnt by the DNN simultaneously. If the observations from these distributions are trained on sequentially, then when learning each task the gradient descent will decrease in a direction that favours the observations from the latest task only. This makes the DNN parameters optimised for the latest task from which observations came from, causing *forgetting* or *catastrophic forgetting* of previous knowledge the DNN has obtained from training on previous tasks[3]. Therefore, during training, the observations from all of the tasks are shuffled together to form one whole distribution. So that when training the gradient will decrease in a direction that optimises the DNN parameters to all of the tasks to be learnt. Similarly during retraining, optimising for observations from a new task during operation causes the learnt distribution to deviate to fit the new observations only. This presents the main challenge when adapting to the PoL during operation.

Whilst new tasks arise sequentially in the changing PoL of an autonomous system, it is not resourcefully efficient to keep all of the observations used during offline training on board to shuffle with and do full retraining every time a new task is encountered. Especially if training on original tasks may take several hours or days to complete. In an autonomous system with limited on-board resources and a continuously changing operational environment, forgetting is a must to make available resources to learn new tasks. In which case forgetting needs to be controlled based on some set of requirements.

Additionally, a generalised autonomous system would have a very wide spectrum of observations used in the offline initial training, which aims at making the trained DNN ready to handle the majority of the scenarios that the system may encounter when deployed in different environments. However, it is not necessary to maintain the same level of performance in all operational environments for all the learnt (original) tasks at deployment time. It is more acceptable for the DNN classifier to sacrifice some performance on tasks not related to the operation environment in order to enhance its performance on PoL tasks, which would increase the system's reliability in its current operational environment.

With these considerations in mind, we develop a retraining pipeline that aims at balancing between the original tasks and the PoL tasks based on some defined criteria.

[3]Forgetting is can be defined as some major or minor decrease in the performance of a DNN on a set of tasks. Catastrophic forgetting, on the other hand, is a drastic decrease in the performance of all learnt tasks.

B. Controlled forgetting retraining pipeline

In this section, we introduce our retraining pipeline and discuss how it tackles the challenges discussed previously to allow for learning PoL tasks, whilst controlling the forgetting of original tasks.

Figure 2 shows the overall retraining pipeline representing two independent training cycles. The first is the initial training, where the system is trained on the original dataset S_0 to produce the initial DNN model M_0, which a system is deployed with. The original dataset represents the observations that were gathered or were available for training on the tasks that developers expected to be what the system will encounter based on a set of requirements and specifications of the system. The second is the continual retraining cycle of the accumulated PoL tasks, this cycle is continuous and occurs whenever the system encounters a new instance.

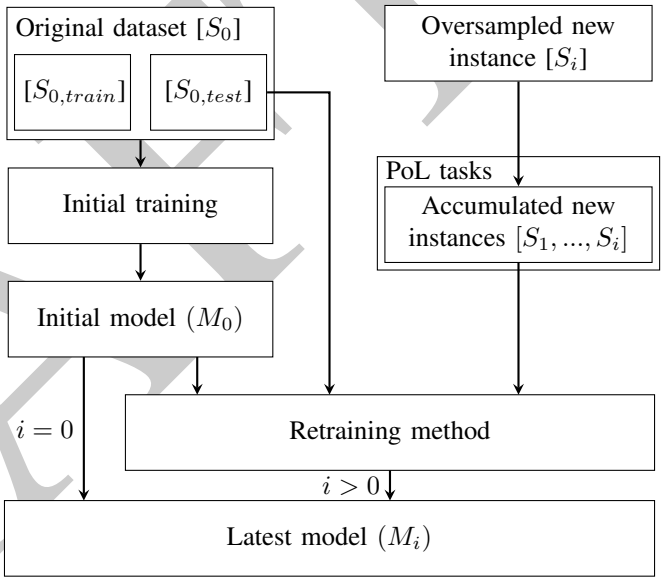


Fig. 2: Retraining Pipeline

In order for the system to learn a new instance, there need to be enough observations of the new instance to allow the SGD to optimise for it i.e. complying with assumption 1. Hence the first step in learning a new instance is oversampling. Oversampling is sampling the input data at a higher frequency than is normally used to run the processes in a system. For example, if the classifier is receiving a video feed then more snap shots will be taken of a certain time interval where the new instance exists, and this will form the observations used to learn the new instance. In another case where the input to the system is a single image, a method of creating different observations from a single new instance will be required. This can be achieved through data augmentation techniques[4], where small image rotations and shifts in the image frame are taken to create oversamples of the new instance.

[4]Data augmentation are techniques used to enhance the size and quality of observations by adding slightly modified versions of the existing observations or by creating new observations inferred from existing observations [34].

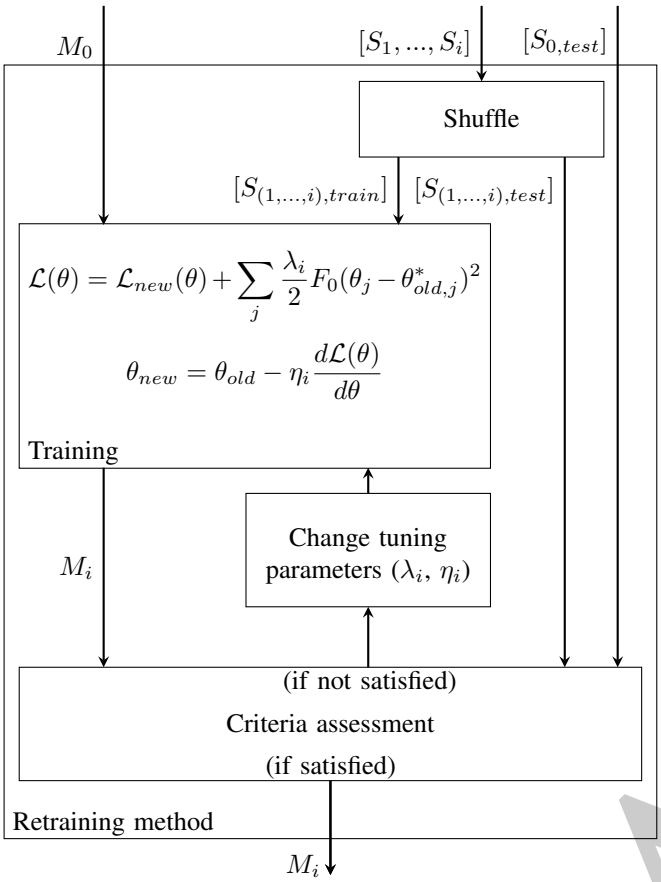


Fig. 3: Retraining Method

The oversampled new instance S_i gets appended to the accumulated new instances storage that represents the observations for the PoL tasks encountered during operation. This then gets shuffled to become partly the retraining dataset and partly the test dataset for evaluating the retrained (latest) model, M_i, on the PoL tasks.

At first instance, it can be thought why not learn new PoL tasks sequentially and independently instead of sequentially and accumulated as we present in this paper? In other words why does the retraining dataset not contain only observations of the last oversampled new instance and the previous PoL tasks get represented as an EWC approximation using an additional Fisher matrix accompanied with an additional hyper-parameter (λ) to tune? This has many challenges and may violate several of the assumptions established previously. It cannot be guaranteed that the latest DNN model will be at a local minimum for all tasks represented by Fisher matrices, hence the validity of using a Fisher matrix to represent each of the previous PoL tasks might not always hold during operation. Attempting to ensure all tasks represented by a Fisher are at a local minimum makes the tuning of the hyper-parameters i.e. the λs extremely computationally expensive. The use of an EWC approximation or more specifically a Fisher matrix to represent a single new task i.e. a distribution that represents a minor subset of the overall operational environment distribution can be debated to violate assumption 3, since not enough observations were provided to represent an approximation of the overall distribution of the tasks in the operational environment but rather of only a single task. Given the discussed limitations of learning PoL tasks independently as they are encountered, we progressed with using accumulated PoL tasks for retraining. Learning sequentially from accumulated PoL would also result in retaining better performance of PoL tasks, as SGD is designed to optimise using random samples from the overall distribution as per assumption 1.

The accumulated PoL tasks $[S_1, ..., S_i]$, the test dataset from the original tasks $[S_{0,test}]$, and the initial trained DNN model, M_0, are fed into the retraining method, which retrains to create the latest updated model (M_i). A full representation of the retraining method is shown in Figure 3. Retraining uses the accumulated PoL tasks to optimise the model. Part of PoL tasks observations are left as a test dataset for evaluation of the retrained model. The starting model used in retraining at any i^{th} retrain, is always the initial model M_0. This makes sure that assumption 2 relating to the DNN used in retraining is at a local minimum for the original tasks, hence ensuring the EWC approximation is valid to use. In the training the mean square error is used to represent the loss function $\mathcal{L}_{new}(\theta)$ for the PoL tasks, this can be replaced by other loss functions, whilst the original tasks are represented using the EWC approximation.

As was described earlier in section II, there are two tuning parameters that can control the forgetting. The λ represents the importance of the original tasks compared to the PoL tasks during retraining, the lower λ is the less priority it is given during retraining, and if it is set to 0 then the original task will not be conserved using EWC. The other parameter is η which can be thought as it constraints how much the model being trained is allowed to deviate from its initial state, the lower η is the more constrained the model is, if η is 0 then the model will not change at all i.e. the PoL tasks will not be learnt.

Why do we need both of these tuning parameters? The SGD hyperparameter η alone is not capable of optimising the retrained model alone on new instances without resulting in significant forgetting as was discussed previously and will be shown in the results. Relying on only tuning λ to meet the criteria assessment can result in the retraining entering a state of insatiability where catastrophic forgetting will occur. This happens due to the multiplication between λ and η that occurs when optimising the combined loss function $\mathcal{L}(\theta)$ by Equation 4. If λ is big, this can result in the retraining steps taken by the SGD optimiser to be too large thus causing the model to diverge instead of converging to a local minimum. Therefore an interplay of the two parameters is needed to allow for continuously stable training.

The criteria assessment block takes the retrained model M_i and test it using the test datasets for the original tasks ($S_{0,test}$) and for the PoL tasks ($S_{(1,...,i),test}$). If the output matches the criteria set by the developers then the model M_i is progressed to be used during operation, else the tuning parameters are tweaked to reach a retrained model that satisfies the criteria.

The criteria itself for learning during operation can be for example that the system should try and maximise its performance at the PoL tasks whilst maintaining a performance of at least 90% on the original tasks, to perhaps meet some regulatory conditions that a system was proved to have at deployment time.

IV. Experimentation

A. Datasets

We use the popular 0-9 hand-written digits MNIST dataset [23], divided into 10 classes, as the original dataset S_0 representing the original tasks used in initial training. This resembles the *expected* operational environment of an autonomous system, that was used at design time. The evaluation of the method is done using the n-MNIST [25] dataset to represent the *actual* operational environment of the DNN classifier, which will be used to obtain random new instances for retraining. The n-MNIST dataset has a similar pattern to the MNIST dataset but with three different types of noise applied to its tasks: Gaussian noise, blur, and reduced contrast with Gaussian noise. This makes it suitable to represent the operational environment of an autonomous system, where patterns of PoL tasks are still similar to original tasks, but may have variations or noise that causes misclassifications.

B. Implementation of retraining pipeline

1) Introduction of new instances: To present a new instance and its oversamples that a DNN classifier may encounter during operation, an observation (instance) is selected randomly from the n-MNIST dataset and augmented using small rotations and shifts. This was achieved using the image augmentation API provided by Keras[5]. The oversamples are split into 1000 training and 250 testing samples for each instance. The test samples of the instance get evaluated on the latest retrained model (M_i). If the instance gets misclassified i.e. accuracy is below the classification threshold, which is 50%, the instance counts as new and needs retraining. Otherwise, the process is repeated for another random instance until a new instance is found. The new instance gets added to the accumulated new instances storage, shuffled and then the DNN classifier is retrained. This process is repeated until the number of retraining episodes needed for the experiment is achieved.

2) DNN Classifier training and retraining pipeline: A convolutional neural network (CNN) with three layers of convolutions and two fully connected layers was used to build the DNN classifier used in the experimentation. These specifications of the DNN were chosen based on its good performance on the MNIST dataset shown by other researchers[6]. An implementation of the EWC algorithm using Pytorch 1.8.1 was adopted from the ContinualAI GitHub repository[7], modified and added it to the retraining pipeline implementation.

[5]https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/\image/ImageDataGenerator

[6]Example: https://github.com/ContinualAI/colab/blob/master/notebooks

[7]https://github.com/ContinualAI/colab

Algorithm 1 Assessment criteria

original$_{accuracy} = []$
for task$_{test}$ in $S_{0,test}$ **do**
  diff = threshold − accuracy$_{M_i}$(task$_{test}$)
  if diff > 0 **then**
    original$_{accuracy}$.append(diff)
  end if
end for
score$_{original}$ = sum(original$_{accuracy}$)
score$_{PoL} = 100 -$ accuracy$_{M_i}$($S_{(1,\ldots,i),test}$)
RetrainingScore = score$_{original}$ + score$_{PoL}$

The assessment criteria used in evaluating the performance of the retraining is depicted by Algorithm 1. The criteria used aims at outputting a *retraining score*, the smaller this score is the more the criteria is satisfied. When the retraining score becomes zero then the criteria is fully satisfied. The *retraining score* is the sum of the *original tasks score* and the *PoL tasks score*. The original tasks score approaches zero as the accuracy of the retrained model on each of the original tasks becomes equal or more than the *controlled forgetting threshold*. This is a threshold that is set to control how much accuracy on the original tasks the system is allowed sacrifice during retraining. The PoL tasks score on the other hand becomes zero as the classification accuracy on the accumulated new instances test dataset ($S_{1,\ldots,i),test}$) reaches 100%.

Hyperopt[8][9] [40]. is used to o identify appropriate values for the hyperparameters λ_i and η_i to maximise the criteria satisfaction. The full code for the experimentation and results are available on: https://github.com/Abanoub-G/PoL_Adptation_EWC. Experiments were run on a computer with an i7-10750H with 16GB RAM and an NVIDIA GeForce RTX 3060 graphics card, running Ubuntu 20.04.02 LTS.

C. Results and discussion

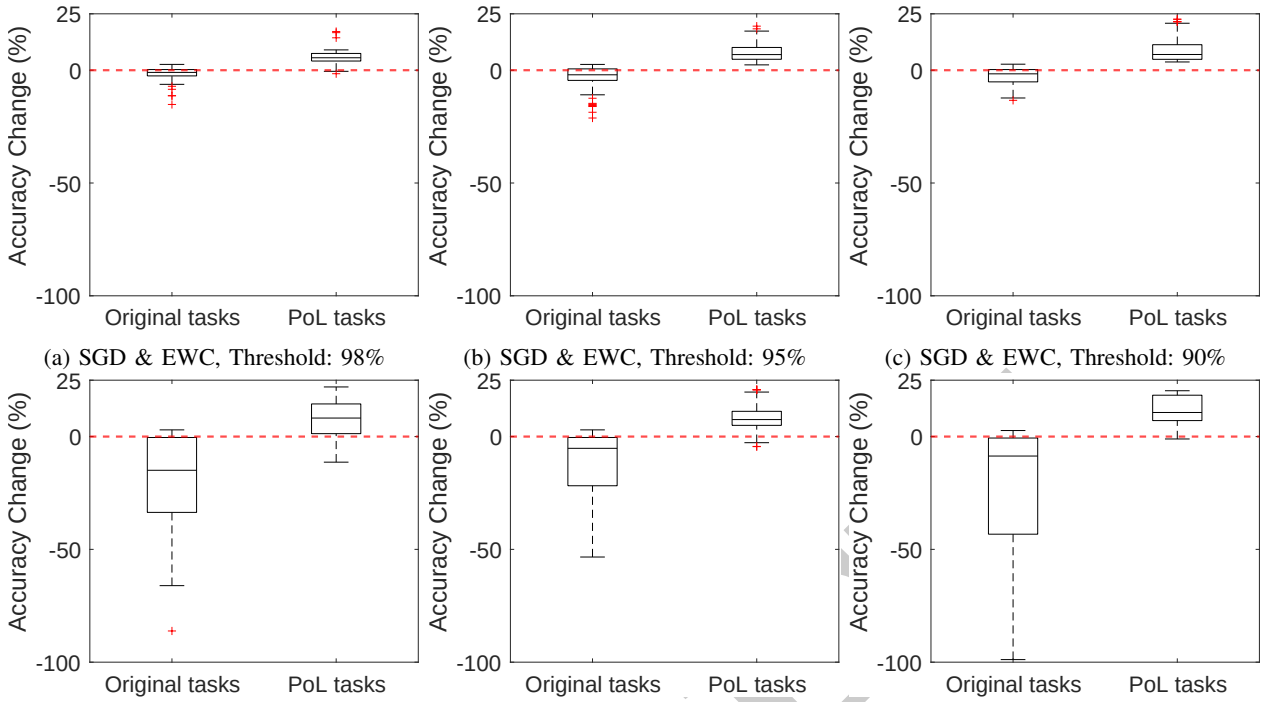
The results presented in this section are for 10 retraining episodes.

1) EWC+SGD vs SGD alone: The results comparing between EWC+SGD and SGD only are presented in Figure 4, where the red dotted line refers to zero change in accuracy from the first model's M_0 accuracy on any given task.

The original tasks accuracy is assessed using the test samples available for each task in the MNIST dataset. Similarly, the PoL tasks are assessed using the test dataset observations for each task provided by the n-MNIST dataset. That is to say, the PoL tasks accuracy shown by the box plots, is not a measure of the retrained model's accuracy on the test dataset from the limited oversampled observations used in retraining,

[8]Hyperopt is a hyperparameters tuning algorithm based on Bayesian optimisation [35, 36, 37].

[9]There are other optimisation algorithms that can be investigated e.g. Hyperband [38], BOHB [39]), to see which maximises the efficiency of the retraining, however we focus mainly on the classification accuracy improvements and control between the PoL tasks and the original tasks gained when EWC is used compared to its absence.



(a) SGD & EWC, Threshold: 98%
(b) SGD & EWC, Threshold: 95%
(c) SGD & EWC, Threshold: 90%
(d) SGD only, Threshold: 98%
(e) SGD only, Threshold: 95%
(f) SGD only, Threshold: 90%

Fig. 4: Spread of retrained model M_i accuracy change on original and PoL tasks during 10 retraining episodes. Shown for EWC + SGD (i.e. tuning λ and η) and SGD only (i.e. $\lambda = 0$ and tuning η) for controlled forgetting thresholds of 98%, 95% and 90%.

but a measure of the generalised accuracy on the overall task. This generalised test dataset obtained from n-MNIST would not practically be accessible to an autonomous system during operation. However, it is used here to assess the general effectiveness of using EWC when retraining on new instances. i.e. focusing on answering **research question 1**.

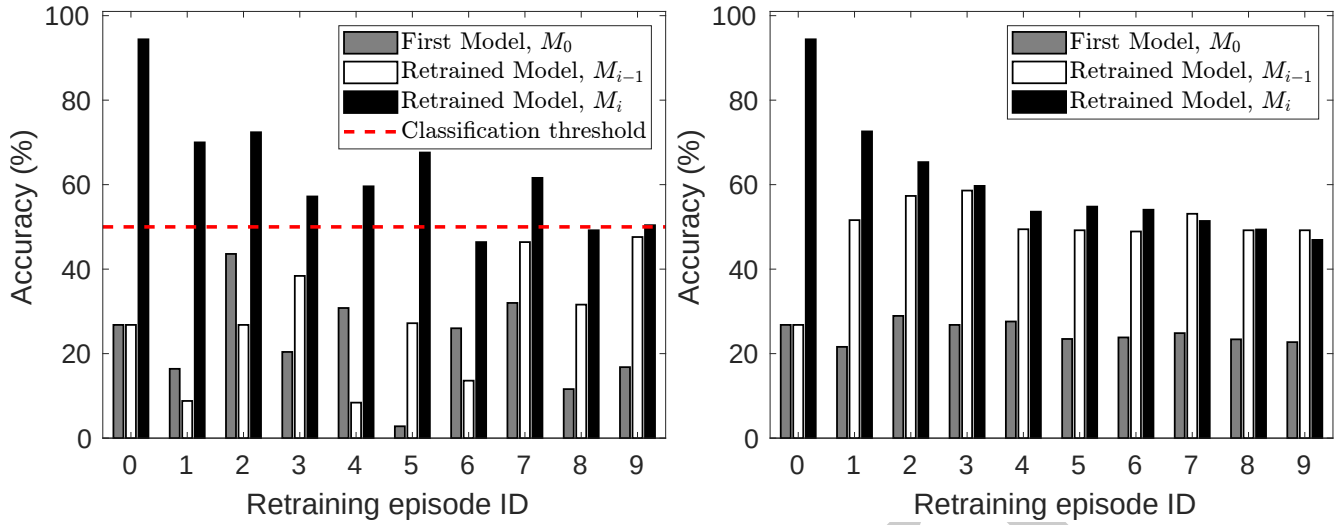
The plots show the spread of the retrained model M_i accuracy change on original and PoL tasks during the 10 retraining episodes, for three controlled forgetting thresholds: 98%, 95% and 90%. The box denotes the interquartile range of accuracy change , non-outlier limits by the whiskers and a horizontal bar for the median.

For all three thresholds used in experimentation, the EWC+SGD shows a much more controlled forgetting paradigm on the original tasks compared to SGD only. This is evident by the very low spread of accuracy change close to the red dotted lines for the EWC+SGD and the very sparse accuracy change below the red dotted line for the SGD only. This shows that the SGD only approach results in significant model drift and even catastrophic forgetting of some tasks, even at very high controlled forgetting thresholds (see Figure 4d).

For both EWC+SGD and SGD only the PoL tasks seem to generally improve in accuracy. The SGD only achieves slightly higher performance but more sparsed on PoL tasks compared of EWC+SGD. This may have merits depending on applications but also indicates overfitting to PoL tasks which is actually the case since the forgetting is much more significant when using SGD only.

2) Performance gain and generalisation for autonomous systems adaptation: Figure 5 shows the before and after retraining accuracy on (a) the latest added new instance in the retraining episode and the (b)accumulated instances. The grey bar is the classification accuracy of the first model M_0. The white bar is the performance of the before last retrained model M_{i-1} whilst the black bar is the accuracy of the latest retrained model M_i. Overall the retraining is causing the classification accuracy to increase on the individual instances and on the overall accumulated instances compared to the first model. Even though some instances do not pass the classification threshold like at retraining episode 6 in Figure 5a, which means they are still misclassified as instances but it still has a positive impact on the classifiers generalisation on the task, as can be seen in Figure 6.

Figure 6 gives a more insightful look at the generalisation on the PoL tasks of the system. This matrix diagram shows a subplot for each of the ten original tasks (learnt from MNIST) and the thirty potential PoL tasks (provided by n-MNIST) that may become part of the system's operational environment. The classifier accuracy on a task from noise 0, 1 or 2 only becomes of importance after an instance from that task appears in the system's operational environment. This indicates that the task has become part of the system's operational environment. The appearance of the instances is denoted by the red dot, showing



(a) Added Instance

(b) Accumulated Instances

Fig. 5: Shows the classification accuracy of the first model M_0 from initial training, before last retrained model M_{i-1} and the latest retraining model M_i on (a) the added new instance S_i and (b) the accumulated instances $[S_1, ...S_i]$. The data presented here is for retraining with controlled threshold of 90% on original tasks.

at which retraining episode an instance from that task was used in retraining.

The evaluation plotted on these subplots is done using the test datasets provided by MNIST and n-MNISt for each of their tasks i.e. similar to the box plots shown earlier, the test datasets are not from the accumulated instances dataset that was used in retraining. This data usually is not available for a system encountering instances of new tasks during operation, but is used here to study how the system generalises on other observations coming from the same task.

As demonstrated by Figure 6, after the DNN classifier is retrained on a new instance the accuracy of the task it comes from improves beyond the classifiers capability on the same task using the first model M_0. This is achieved at the expense of some forgetting of the original tasks, which is acceptable given that the model is drifting to fit its PoL.

It is challenging to assess the generalisation of the DNN classifier on new tasks just by evaluating its accuracy on the available observations, as was demonstrated by the earlier discussion on Figures 5 and 6. This may introduce limitations on the system's capability of monitoring and assessing its generalisation behaviour on new tasks from a few new instances. However, monitoring the improvement on new instances classification can be an indicator of good or bad generalisation. For example improved accuracy can be a positive sign, but achieving really high accuracy on all of the new instances can be a sign of overfitting and thus poor generalisation.

3) Retraining speed: Another obvious limitation of the current retraining pipeline is that as more new instances appear the retraining process will become slower as there are more observations to optimise the model for in the accumulated new instances storage. This disadvantage may be limited by decreasing oversamples of the instances in the accumulated new instances storage used in retraining as the number of new instances increase. Removal of instances completely from the storage that does not fall into the PoL of the system any more can be another approach for tackling this limitation. A way of achieving that can be by keeping track of how often the system encounters similar instances (perhaps by using dissimilarity measures [7] to detect them), and decide whether it is beneficial to remove or keep certain instances based on some criteria put by developers or regulators. If the system is encountering a large number of new instances that all fall in its PoL then that would invite for developers to investigate the observations used in initial training, as the expected and actual operational environments are very different.

4) A Note on transfer learning: Learning from as little as one instance is generally challenging and does not abide with current machine learning paradigms, which relies on the availability of abundant observations to learn new tasks. Using oversampling and data augmentation techniques, as done in the experimentation, certainly improves learning of the new tasks and its generalisation. However a major contribution to the success of the methods is caused by transfer learning obtained from original tasks. This may limit the use of the proposed approach from learning tasks following a completely different pattern, e.g Permuted-MNIST [11] and Fashion-MNIST [24]. Never the less, this is acceptable, as the method is really aimed at increasing the reliability of autonomous systems on tasks in its operational environment that have some variations such as a novel type of noise rather than a completely novel pattern.

V. Conclusions

In this paper, we have demonstrated that using EWC allows for more controlled learning of new tasks from new instances whilst controlling the limit allowed for forgetting of original

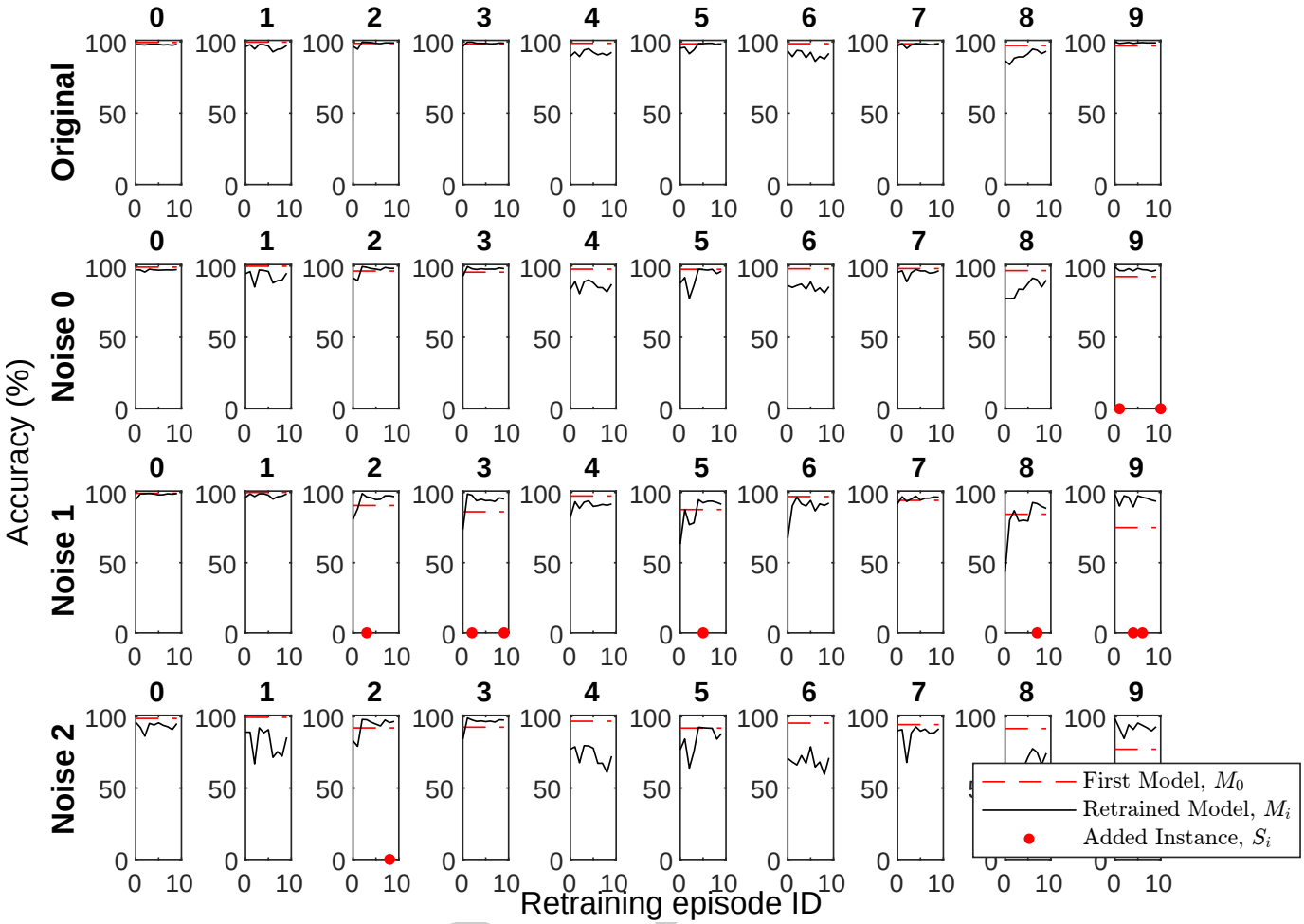


Fig. 6: Matrix plot showing accuracy change during retraining episodes, for controlled forgetting threshold of 90%, on each of the original tasks and the possible PoL tasks from the n-MNIST dataset. The red dots shows the instances that were added and at which retraining episode they were added from the ten retraining episodes.

tasks. This was followed by studying the gains in accuracy and generalisation of retrained DNN classifiers on new tasks using the MNIST and the n-MNIST datasets to represent expected and actual operational environments, respectively. Thus gain an understanding of using this method for pattern of life adaption of autonomous systems. The results show that there are clear signs of generalisation on pattern of life tasks from retraining on limited observations of new instances derived from these pattern of life tasks.

VI. Future works

Posing questions for future works to allow for the adoption of this retraining pipeline in real-time autonomous systems include:

- Developing runtime monitors to help an autonomous system evaluate its performance on PoL tasks after retrains inferred from the classifiers accuracy improvement on encountered new instances.
- Investigate methods for maintaining the retraining speed of the DNN classifier as the new instances storage increases to still allow for efficient retraining.

Acknowledgments

This research is part of an iCASE PhD funded by EPSRC and Thales UK. Special thanks to —ADD HERE NAMES– for helping with reviewing the paper.

References

[1] N. Kalra and S. M. Paddock. *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* Santa Monica, CA: RAND Corporation, 2016.

[2] G. Chance et al. "An agency-directed approach to test generation for simulation-based autonomous vehicle verification". In: *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE. 2020, pp. 31–38.

[3] K. I. Eder, W.-l. Huang, and J. Peleska. "Complete Agent-driven Model-based System Testing for Autonomous Systems". In: *arXiv preprint arXiv:2110.12586* (2021).

[4] R. Craddock, D. Watson, and W. Saunders. "Generic Pattern of Life and behaviour analysis". In: *2016 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support, CogSIMA 2016* (2016), pp. 152–158.

[5] P. Koopman and M. Wagner. *Positive Trust Balance for Self-driving Car Deployment*. Vol. 2. Springer International Publishing, 2020, pp. 351–357. URL: http://dx.doi.org/10.1007/978-3-030-55583-2_26.

[6] C. Harper et al. "Safety Validation of Autonomous Vehicles using Assertion-based Oracles". In: *arXiv preprint arXiv:2111.04611* (2021).

[7] D. Hond, H. Asgari, and D. Jeffery. "Verifying Artificial Neural Network Classifier Performance Using Dataset Dissimilarity Measures". In: *Proceedings - 19th IEEE International Conference on Machine Learning and Applications, ICMLA 2020* (2020), pp. 115–121.

[8] J. D. Schaffer and W. H. Land. "Predicting with Confidence: Classifiers that Know What They Don't Know". In: *Procedia Computer Science* 114 (2017), pp. 200–207. URL: https://doi.org/10.1016/j.procs.2017.09.061.

[9] A. Mandelbaum and D. Weinshall. "Distance-based Confidence Score for Neural Network Classifiers". In: (2017). arXiv: 1709.09844. URL: http://arxiv.org/abs/1709.09844.

[10] C. Xing et al. "Distance-Based Learning from Errors for Confidence Calibration". In: (2019), pp. 1–12. arXiv: 1912.01730. URL: http://arxiv.org/abs/1912.01730.

[11] I. J. Goodfellow et al. "An empirical investigation of catastrophic forgetting in gradient-based neural networks". In: *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings* (2014). arXiv: 1312.6211.

[12] J. Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences of the United States of America* 114.13 (2017), pp. 3521–3526. arXiv: 1612.00796.

[13] E. T. Barr et al. "The oracle problem in software testing: A survey". In: *IEEE Transactions on Software Engineering* 41.5 (2015), pp. 507–525.

[14] J. M. Zhang et al. "Machine Learning Testing: Survey, Landscapes and Horizons". In: *IEEE Transactions on Software Engineering* (2020), pp. 1–1. arXiv: 1906.10742.

[15] A. Stocco and P. Tonella. "Towards Anomaly Detectors that Learn Continuously". In: *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, Oct. 2020, pp. 201–208. URL: https://ieeexplore.ieee.org/document/9307667/.

[16] Y. X. Wang, D. Ramanan, and M. Hebert. "Growing a brain: Fine-tuning by increasing model capacity". In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-January (2017), pp. 3029–3038. arXiv: 1907.07844.

[17] Z. Li and D. Hoiem. "Learning without Forgetting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (2018), pp. 2935–2947. arXiv: 1606.09282.

[18] F. Zenke, B. Poole, and S. Ganguli. "Continual learning through synaptic intelligence". In: *34th International Conference on Machine Learning, ICML 2017* 8 (2017), pp. 6072–6082. arXiv: 1703.04200.

[19] D. Maltoni and V. Lomonaco. "Continuous learning in single-incremental-task scenarios". In: *Neural Networks* 116 (2019), pp. 56–73. arXiv: 1806.08568. URL: https://doi.org/10.1016/j.neunet.2019.03.010.

[20] J. M. Allred and K. Roy. "Controlled Forgetting: Targeted Stimulation and Dopaminergic Plasticity Modulation for Unsupervised Lifelong Learning in Spiking Neural Networks". In: *Frontiers in Neuroscience* 14.January (2020), pp. 1–16. arXiv: 1902.03187.

[21] G. I. Parisi et al. "Continual lifelong learning with neural networks: A review". In: *Neural Networks* 113 (2019), pp. 54–71. arXiv: 1802.07569. URL: https://doi.org/10.1016/j.neunet.2019.01.012.

[22] R. Kemker et al. "Measuring catastrophic forgetting in neural networks". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (2018), pp. 3390–3398. arXiv: 1708.02072.

[23] L. Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[24] H. Xiao, K. Rasul, and R. Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: (2017), pp. 1–6. arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747.

[25] S. Basu et al. "Learning Sparse Feature Representations Using Probabilistic Quadtrees and Deep Belief Nets". In: *Neural Processing Letters* 45.3 (2017), pp. 855–867. arXiv: 1509.03413.

[26] X. Huang et al. *A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability*. 2020. arXiv: 1812.08342. URL: http://arxiv.org/abs/1812.08342.

[27] I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[28] T. Parr and J. Howard. "The Matrix Calculus You Need For Deep Learning". In: 2018 (2018), pp. 1–33. arXiv: 1802.01528. URL: http://arxiv.org/abs/1802.01528.

[29] F. Huszár. "Note on the quadratic penalties in elastic weight consolidation". In: *Proceedings of the National Academy of Sciences of the United States of America* 115.11 (2018), E2496–E2497. arXiv: arXiv:1712.03847v1.

[30] D. J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. USA: Cambridge University Press, 2002.

[31] S. M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory*. 1993, p. 180.

[32] A. Ly et al. "A Tutorial on Fisher information". In: *Journal of Mathematical Psychology* 80 (2017), pp. 40–55. arXiv: 1705.01064.

[33] J. Martens. "New insights and perspectives on the natural gradient method". In: *Journal of Machine Learning Research* 21 (2020), pp. 1–76. arXiv: 1412.1193.

[34] C. Shorten and T. M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (2019). URL: https://doi.org/10.1186/s40537-019-0197-0.

[35] J. Mockus. "On the Bayes Methods for Seeking the Extremal Point". In: *IFAC Proceedings Volumes* 8.1 (1975), pp. 428–431. URL: http://dx.doi.org/10.1016/S1474-6670(17)67769-3.

[36] J. Snoek, H. Larochelle, and R. P. Adams. "Practical Bayesian optimization of machine learning algorithms". In: *Advances in Neural Information Processing Systems* 4 (2012), pp. 2951–2959. arXiv: 1206.2944.

[37] M. Feurer and F. Hutter. *Hyperparameter Optimization*. 2019, pp. 3–33.

[38] L. Li et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *Journal of Machine Learning Research* 18 (2018), pp. 1–52. arXiv: 1603.06560.

[39] S. Falkner, A. Klein, and F. Hutter. "BOHB: Robust and Efficient Hyperparameter Optimization at Scale". In: *35th International Conference on Machine Learning, ICML 2018* 4 (2018), pp. 2323–2341. arXiv: 1807.01774.

[40] J. Bergstra, D. Yamins, and D. Cox. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 115–123. URL: https://proceedings.mlr.press/v28/bergstra13.html.