Dynamic Continual Learning: Harnessing Parameter Uncertainty for Improved Network Adaptation

Christopher F. Angelini
Dept. of Electrical and Computer Engineering
Rowan University
Glassboro, New Jersey, USA
angelinic0@rowan.edu

Nidhal C. Bouaynaya
Dept. of Electrical and Computer Engineering
Rowan University
Glassboro, New Jersey, USA
bouaynaya@rowan.edu

***Abstract*—When fine-tuning Deep Neural Networks (DNNs) to new data, DNNs are prone to overwriting network parameters required for task-specific functionality on previously learned tasks, resulting in a loss of performance on those tasks. We propose using parameter-based uncertainty to determine which parameters are relevant to a network's learned function and regularize training to prevent change in these important parameters. We approach this regularization in two ways: (1), we constrain critical parameters from significant changes by associating more critical parameters with lower learning rates, thereby limiting alterations in those parameters; (2), important parameters are restricted from change by imposing a higher regularization weighting, causing parameters to revert to their states prior to the learning of subsequent tasks. We leverage a Bayesian Moment Propagation framework which learns network parameters concurrently with their associated uncertainties while allowing each parameter to contribute uncertainty to the network's predictive distribution, avoiding the pitfalls of existing sampling-based methods. The proposed approach is evaluated for common sequential benchmark datasets and compared to existing published approaches from the Continual Learning community. Ultimately, we show improved Continual Learning performance for Average Test Accuracy and Backward Transfer metrics compared to sampling-based methods and other non-uncertainty-based approaches.**

***Index Terms*—Continual Learning, Deep Variational Inference, Parameter Uncertainty, Moment Propagation**

I. Introduction

The term "Narrow AI" is gaining traction for describing Artificial Intelligence (AI) and Machine Learning (ML) systems that cannot adapt to information after deployment. In standard training schemes, Deep Neural Networks (DNNs) assume the collection of observations a network is trained on will accurately describe the environment in which it is deployed. In the real world, DNNs are regularly subject to out-of-distribution (OOD) data, various types of noise, shifting distributions of conceptual objectives, and may require adaptation to new data after the initial training period [1]. These characteristics of real-world data undermine assumptions of data consistency made during training and require DNNs to have the ability to adapt and expand upon previously learned data representations.

Include this acknowledgment: This work was supported by National Science Foundation (NSF) Awards NSF ECCS 1903466 and NSF OAC 2008690.

Unlike biological learning systems, which can adapt and consolidate learned information at will, standard machine learning systems restrict network performance to the most recently trained task [2], [3]. Fine-tuning networks to new information generally results in a partial or complete loss of performance on previously trained tasks, known as *Catastrophic Forgetting* or *Catastrophic Interference* [4], [5]. This phenomenon arises from overwriting and replacing network parameters during the training process for a new task. As a result, a sub-discipline of machine learning called *Continual Learning* (CL), or *Lifelong Learning*, has emerged, focusing on mitigating Catastrophic Interference in DNNs over a sequence of tasks.

Current approaches to CL can be categorized into four main categories: Regularization, Dynamic Architectures, Rehearsal, and Dual Memory Systems [3]. Regularization-based approaches restrict parameter change when training subsequent tasks attempting to preserve previous network representations [2], [6]–[8]. Dynamic Architecture approaches add neural resources to expand on previous representations [9], [10]. Rehearsal approaches preserve encompassing data samples that describe most information learned from previous tasks, supplementing training new tasks with these selected data samples [11]–[13]. Dual Memory Systems consolidate information from short-term, temporary to long-term, permanent memory to enable rapid adaptation while preserving previously encountered information [14]–[17].

Methods described in this paper will focus on regularization-based approaches to avoid undesirable characteristics of other approaches, such as storing previous data, adding neural resources for new tasks, or complex network and optimization structures. At the limit, allowing for such characteristics would create an ever-growing storage or computational resource requirement unsuitable for many real-world applications. Regularization-based approaches focus on constraining the optimization process when learning a new task to avoid significant changes to parameters learned in previous tasks. Most regularization approaches rely on identifying important parameters within the network and adjusting regularization according to the relevance of each parameter to a previous task. This contrasts with applying a uniform factor across all network parameters.

This paper details two new regularization approaches to CL by managing changes in network parameters with learned parameter uncertainty inherent to Bayesian Deep Learning networks. The learned statistical uncertainty derived from the approximated posterior distribution serves as the basis for determining the importance of each network parameter in the context of a previously learned task. Contrary to existing Bayesian approaches that rely on Monte Carlo sampling to estimate network uncertainty, we developed a network-agnostic framework, called Moment Propagation (MP) [18], to learn the values of network parameters and their associated uncertainty in an online manner, without resorting to computationally expensive sampling techniques. In contrast to previous approaches to deep Variational Inference (VI) [19], [20], MP networks approximates the network's predictive distribution by propagating the first two moments, mean and covariance, through the entire network. The first-order Taylor Series approximation is used to enable propagation of covariance through non-linearities in the network i.e. non-linear activation functions such as ReLU. This approach facilitates the direct maximization of the Evidence Lower Bound Objective without sampling the network likelihood [18].

We leverage parameter uncertainty learned from a MP network to govern our CL process in two ways: Learning Rate Adaptation and Per-parameter Bayesian-based regularization. Our contributions are as follows:

- We demonstrate the MP framework's ability to autonomously identify important network parameters relevant to the trained network function.
- We leverage the learned parameter uncertainties to regularize the training of new tasks, preventing the occurrence of catastrophic forgetting of previous tasks.
- We demonstrate catastrophic forgetting mitigation in the task incremental learning setting with multiple sequential benchmark datasets.
- We compare the results of our approach with state-of-the-art approaches and baseline metrics for each network.

II. Moment Propagation Framework

MP leverages principles from VI to approximate the variational posterior of a network by imposing an approximating distribution, also known as variational distribution, $q_\theta(\mathbf{\Omega})$, over the network parameters. The variational distribution is then optimized by minimizing the Kullback-Leibler (KL) divergence between the variational distribution and the true posterior distribution, $p(\mathbf{\Omega}|\mathcal{D})$, where $\mathcal{D} = \{\mathbf{X}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{n}$, represents the training data with i^{th} input $\mathbf{X}^{(i)}$ and corresponding label $\mathbf{y}^{(i)}$. However, given that minimizing the KL divergence between the variational distribution and the true posterior is intractable due to the log evidence, an equivalent form must be considered. The log evidence can be isolated as it does not depend on the variational distribution nor will it affect the optimization. A tractable and equivalent objective called the Evidence Lower Bound (ELBO) can be maximized in place of the original KL divergence between the variational distribution and the true posterior distribution, shown in Equation (1).

$$\text{ELBO} = \mathbb{E}_{q_\theta(\mathbf{\Omega})}[\ln p(\mathcal{D}|\mathbf{\Omega})] - \text{KL}_{q_\theta(\mathbf{\Omega})}[q_\theta(\mathbf{\Omega})||p(\mathbf{\Omega})] \quad (1)$$

When maximizing the ELBO it's components, the network log-likelihood, $\mathbb{E}_{q_\theta(\mathbf{\Omega})}[\ln p(\mathcal{D}|\mathbf{\Omega})]$, is maximized and the KL divergence between the network prior and the variational distribution, $\text{KL}_{q_\theta(\mathbf{\Omega})}[q_\theta(\mathbf{\Omega})||p(\mathbf{\Omega})]$, is minimized. Existing approaches to VI for DNNs [20] rely on approximating the network's log-likelihood by creating variations in the predictive distribution through Monte Carlo sampling of the approximated variational posterior. A distribution is then fit over these predictions to approximate the predictive distribution, allowing for the estimation of the network log-likelihood and the resulting gradient update of the network parameters. However, this process requires performing inference for each requested sample, increasing the time and computation requirements as the number of samples increase producing a trade-off between network efficacy and efficiency.

This trade-off can be circumvented by allowing parameters to contribute their learned uncertainty to features as they are transmitted through the network, thereby accumulating uncertainty in the network's predictive distribution. We introduced a framework that propagates the first two moments of the predictive distribution through the non-linear layers of the network using a first-order Taylor Series approximation for variance, called *Moment Propagation* (MP) [18]. This method involves learning the mean and variance of network parameters in an online fashion while enabling these parameters to influence the predictive mean and covariance. MP facilitates the propagation of an analytical expression for both the mean and covariance moments, allowing the predictive distribution to be determined without the need for sampling. Consequently, the stochasticity typically associated with the sampling process is removed from the predictive distribution estimation permitting the uncertainty in the predictive distribution to be solely based on uncertainty contributed by the network parameters. The propagation of uncertainty through this method provides an effective, deterministic measure for both mean and covariance, yielding a consistent and repeatable assessment of predictive uncertainty that is directly differentiable. Overall, MP enhances estimates of both posterior and predictive distributions with unbiased, directly differentiable evaluations of the network log-likelihood.

In the following sections, MP is derived for a convolutional neural network with L layers. To streamline notation, the reference to the layer l is excluded from the representation for the l^{th} layer. The derivations for various network layers are presented while assuming the following:

- Without loss of generality, the input feature to the network at layer $l = 0$, convolutional or linear, is treated as deterministic.
- The j^{th} network parameter w_j follows a Normal distribution $w_j \sim \mathcal{N}(\mu_{w_j}, \sigma^2_{w_j})$.
- The network parameters are independent of each other and the input.

1) Propagation through the l^{th} 2D convolutional layer: Let $\mathcal{G} = \mathcal{W} * \mathcal{X} + \boldsymbol{b}$, where, $*$ denotes the convolution operation. For the l^{th} convolutional layer, let $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times ch}$ as a random tensor denoting the layer input, $\mathcal{W} \in \mathbb{R}^{k_1 \times k_2 \times ch \times f}$ as a random tensor denoting the layer weights, $\boldsymbol{b} \in \mathbb{R}^f$ as a random vector of denoting the layer biases, and $\mathcal{G} \in \mathbb{R}^{out_1 \times out_2 \times f}$ as random tensor denoting the propagated feature.

We consider the vectorized form of each filter within weight random tensor $\mathcal{W}$ reforming the random tensor to matrix $\boldsymbol{W} \in \mathbb{R}^{k \cdot k \cdot ch \times f}$. Let $\boldsymbol{W} = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_f]$, where $\boldsymbol{w}_i$ is the i^{th} column of $\boldsymbol{W}$, representing the i^{th} vectorized filter with the mean and covariance $\boldsymbol{\mu}_{\boldsymbol{w}_i} \in \mathbb{R}^{k \cdot k \cdot ch}$ and $\boldsymbol{\Sigma}_{\boldsymbol{w}_i} \in \mathbb{R}^{k \cdot k \cdot ch \times k \cdot k \cdot ch}$, respectively. Similarly, we consider the vectorized form of the image patches under each filter within the input random tensor $\mathcal{X}$ reforming the random tensor to matrix $\boldsymbol{X} \in \mathbb{R}^{k \cdot k \cdot ch \times out_1 \cdot out_2}$. Let $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{[out_1 \cdot out_2]}]$,where $\boldsymbol{x}_o$ is the o^{th} column of $\boldsymbol{X}$, representing the o^{th} vectorized image patch with the mean and covariance $\boldsymbol{\mu}_{\boldsymbol{x}_o} \in \mathbb{R}^{k \cdot k \cdot ch}$ and $\boldsymbol{\Sigma}_{\boldsymbol{x}_o} \in \mathbb{R}^{k \cdot k \cdot ch \times k \cdot k \cdot ch}$, respectively. It follows that the resulting random element $g_{o,i}$ with mean and covariance elements, $\mu_{g_{o,i}}$ and $\sigma_{g_{o,i}}$, contained within the resulting random matrix $\boldsymbol{G}$ can be derived for each filter $i = 1 \cdots f$ and output feature pixel $o = 1 \cdots (out_1 \cdot out_2)$ with the matrix-vector multiplication as shown in Equation (2).

$$\begin{aligned} \mu_{g_{o,i}} &= \boldsymbol{\mu}_{\boldsymbol{w}_i}^T \boldsymbol{\mu}_{\boldsymbol{x}_o} + \mu_{b_i} \\ \sigma_{g_{o,i}}^2 &= \operatorname{tr}(\boldsymbol{\Sigma}_{\boldsymbol{x}_o} \boldsymbol{\Sigma}_{\boldsymbol{w}_i}) + \boldsymbol{\mu}_{\boldsymbol{x}_o}^T \boldsymbol{\Sigma}_{\boldsymbol{w}_i} \boldsymbol{\mu}_{\boldsymbol{x}_o} + \boldsymbol{\mu}_{\boldsymbol{w}_i}^T \boldsymbol{\Sigma}_{\boldsymbol{x}_o} \boldsymbol{\mu}_{\boldsymbol{w}_i} + \sigma_{b_i}^2 \end{aligned} \quad (2)$$

2) Propagation through the k^{th} linear layer: Let $\boldsymbol{z} = \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}$, where, for the k^{th} layer, $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ is a random matrix of weights, $\boldsymbol{b} \in \mathbb{R}^m$ is a random vector of biases, and $\boldsymbol{z} \in \mathbb{R}^m$ is the resulting random vector. Let $\boldsymbol{W} = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_m]$, where $\boldsymbol{w}_i$ is the i^{th} column of $\boldsymbol{W}$, with the mean and covariance of $\boldsymbol{w}_i$ represented as $\boldsymbol{\mu}_{w_i} \in \mathbb{R}^n$ and $\boldsymbol{\Sigma}_{w_i} \in \mathbb{R}^{n \times n}$. The random input vector is represented by $\boldsymbol{x} \in \mathbb{R}^n$ with mean and variance value vectors $\boldsymbol{\mu}_{\boldsymbol{x}} \in \mathbb{R}^n$ and $\boldsymbol{\Sigma}_{\boldsymbol{x}} \in \mathbb{R}^{n \times n}$, respectively. It follows that the mean and variance elements, μ_{z_i} and $\sigma_{z_i}^2$, contained within the resulting random vector $\boldsymbol{z}$ can be derived for elements $i = 1 \cdots m$, with the matrix-vector multiplication as shown in Equation (3).

$$\begin{aligned} \mu_{z_i} &= \boldsymbol{\mu}_{w_i}^T \boldsymbol{\mu}_x + \mu_{b_i} \\ \sigma_{z_i}^2 &= \operatorname{tr}\left(\boldsymbol{\Sigma}_x \boldsymbol{\Sigma}_{w_i}\right) + \boldsymbol{\mu}_x^T \boldsymbol{\Sigma}_{w_i} \boldsymbol{\mu}_x + \boldsymbol{\mu}_{w_i}^T \boldsymbol{\Sigma}_x \boldsymbol{\mu}_{w_i} + \sigma_{b_i}^2 \end{aligned} \quad (3)$$

3) Propagation through p^{th} Batch Normalization: Let $\boldsymbol{F} = \boldsymbol{\gamma} \odot \hat{\boldsymbol{X}} + \boldsymbol{\beta}$, where $\hat{\boldsymbol{X}}$ is the normalized input $\boldsymbol{X}$ according to Equation 4. Let input $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_B]$ and $\boldsymbol{x}_b$ is the b^{th} batched random vector of $\boldsymbol{X}$. Each random input vector is represented by $\boldsymbol{x_b} \in \mathbb{R}^n$ with mean and variance value vectors $\boldsymbol{\mu}_{\boldsymbol{x}_b} \in \mathbb{R}^n$ and $\boldsymbol{\Sigma}_{\boldsymbol{x}_b} \in \mathbb{R}^{n \times n}$, respectively. Each input vector is normalized according to Equation 4, where $\boldsymbol{\mu_N} \in \mathbb{R}^{\boldsymbol{n}}$ and $\boldsymbol{\sigma_N^2} \in \mathbb{R}^{\boldsymbol{n}}$ represent the feature-wise (channel-wise in the case of 2D batch normalization) mean and variance over all $\boldsymbol{\mu}_{\boldsymbol{x}_b}$. The normalized moment vectors of mean and variance are represented by $\boldsymbol{\mu}_{\hat{x}_b}$ and $\boldsymbol{\Sigma}_{\hat{x}_b}$, correspond to $\boldsymbol{x}_b$ in $\hat{\boldsymbol{X}} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_B]$. A small value, ϵ is added to the denominators for numerical stability.

$$\boldsymbol{\mu}_{\hat{x}_b} = \frac{\boldsymbol{\mu_{x_b}} - \boldsymbol{\mu_N}}{\sqrt{\boldsymbol{\sigma_N^2} + \epsilon}} \qquad \boldsymbol{\Sigma}_{\hat{x}_b} = \frac{\boldsymbol{\Sigma}_{x_b}}{\boldsymbol{\sigma_N^2} + \epsilon} \quad (4)$$

4) Propagation through a non-linear activation: Let $\mathcal{Z} = \Psi(\mathcal{G})$ represent some non-linear activation function (e.g. ReLU, Hyperbolic Tangent, Softmax) of a random vector input $\mathcal{G} \in \mathbb{R}^{m \times m \times ch}$ with mean $\boldsymbol{\mu}_z$ and covariance $\boldsymbol{\Sigma}_z$. The mean and covariance for the resulting random vector $\boldsymbol{g}$ can be approximated using the first-order Taylor series approximation [18] in Equation (5) where $\odot$ is the element-wise product of the incoming covariance matrix, $\boldsymbol{\Sigma}_z$, and the squared gradient, ∇, of non-linear function with respect to the incoming mean, $\boldsymbol{\mu}_z$.

$$\begin{aligned} \boldsymbol{\mu}_g &\approx \Psi(\boldsymbol{\mu}_z) \\ \boldsymbol{\Sigma}_g &\approx \boldsymbol{\Sigma}_z \odot \nabla\Psi(\boldsymbol{\mu}_z) \nabla\Psi(\boldsymbol{\mu}_z)^T \end{aligned} \quad (5)$$

For the Softmax classification layer at the output of the network, let $\boldsymbol{\mu}_{\hat{\boldsymbol{y}}}$ and $\boldsymbol{\sigma}_{\hat{\boldsymbol{y}}}^2$ denote the mean and variance, respectively, that will be used to infer the ELBO objective function.

A. Closed Form ELBO

Using the propagated values $\boldsymbol{\mu}_{\hat{\boldsymbol{y}}}$ and $\boldsymbol{\sigma}_{\hat{\boldsymbol{y}}}^2$, the ELBO, Equation (1), can be written in closed form, as shown in Equation (6). The weighting variable $\boldsymbol{\tau}$ is added to control the level of explicit regularization toward the prior induced by the KL divergence term. The KL divergence is computed on a per-parameter basis and then summed, where $|\boldsymbol{\Omega}|$ denotes the cardinality of the set $\boldsymbol{\Omega}$, i.e., the number of weight parameters, and N denotes the number of classes or output nodes of the final layer.

B. Training Moment Propagation

Contrary to other traditional probabilistic problems where VI can be applied, estimating a prior for a DNN is difficult before any training has occurred. Given the dual objective of the ELBO, reverting to a prior far from the parameterization of the true posterior would be detrimental to the learning process. Instead, sparsity-inducing priors, such as the standard normal, $p(\Omega_i) = \mathcal{N}\left(\mu_{p_i} = 0, \sigma_{p_i}^2 = 1\right)$, can be used to remove parameters that are not required to maximize the model likelihood by reverting their mean and variance, to that of the sparsity inducing prior.

Thus, the maximum variance of a parameter present in the network will be $\sigma_{\omega_i}^2 = 1$. The incorporation of a sparsity-inducing prior in the ELBO objective aligns with the Minimum Description Length principle. This is achieved by inherently reducing the complexity of a DNN, specifically by systematically eliminating parameters that are not necessary for a given task [21], [22].

$$\text{ELBO} = -\frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln\left(|\boldsymbol{\Sigma}_{\hat{\boldsymbol{y}}}|\right) - \frac{1}{2}\left((\boldsymbol{y} - \boldsymbol{\mu}_{\hat{\boldsymbol{y}}})^T {\boldsymbol{\Sigma}_{\hat{\boldsymbol{y}}}}^{-1} (\boldsymbol{y} - \boldsymbol{\mu}_{\hat{\boldsymbol{y}}})\right) - \frac{\tau}{2}\sum_{i=1}^{|\boldsymbol{\Omega}|}\left(-1 + \frac{(\mu_{q_{\omega_i}} - \mu_{p_i})^2}{\sigma_{q_{\omega_i}}^2} + \ln\left(\frac{\sigma_{p_i}^2}{\sigma_{q_{\omega_i}}^2}\right) + \frac{\sigma_{q_{\omega_i}}^2}{\sigma_{p_i}^2}\right) \quad (6)$$

III. Weight Uncertainty

After training, parameters essential for a task are expected to exhibit reduced uncertainty. Conversely, parameters unessential for the task are expected to gravitate towards the sparsity-inducing prior, resulting in increased uncertainty. To measure a parameter's relative importance to a DNN's function, two forms of uncertainty are considered: the variance of each parameter, $\sigma^2_{\omega_i}$, and the Signal-to-Noise Ratio (SNR) of each parameter, $\text{SNR}_{\omega_i} = |\mu_{\omega_i}|/\sigma^2_{\omega_i}$. Parameter importance is inversely proportional to a parameter's variance and proportional to a parameter's SNR. MP's ability to self-determine important parameters is demonstrated by observing a cumulative distribution function (CDF) of both parameter SNR and variance from a trained two 800-node hidden layer fully connected network. The CDFs for parameter uncertainty are then correlated to the network's ability to prune parameters based on each parameter's learned uncertainty.

The CDFs for the uncertainty measurements are presented in Figure 1. Parameter importance based on SNR and Variance demonstrates that 95% of the fully connected network parameters are uncertain and, thus, are considered unimportant. Similarly, the CDF of the variance of the parameters demonstrates that 95% of the parameters have been optimized to be equivalent to the prior and are thus uncertain.

Although parameters aligning with the prior may appear equivalent, it doesn't necessarily imply insignificance to the network's functional approximation. To demonstrate parameter importance to a learned network function, parameters are ordered and pruned according to their SNR and variance at various percentages of the total network parameters. The subsequent impact on network performance is then assessed.

During pruning, all network parameters are ordered based on importance, regardless of layer, as there is no guarantee that important parameters will be evenly distributed across all layers. This approach allows for more efficient pruning without compromising performance. The performance of the pruned network, considering SNR and variance-based pruning, is compared against random pruning (the lower performance bound) and pruning based on the smallest absolute value. Additionally, the performance of the MP framework is compared to Bayes-by-Backprop (BBB) and standard deterministic networks, both featuring the same two hidden layer architecture. For MP and BBB frameworks, smallest absolute value-based pruning is performed on parameters using the mean of the parameter's approximating distribution.

Figure 2 shows that the MP network maintained more performance, even with a higher percentage of the network pruned compared to the BBB and deterministic frameworks. Although variance-based pruning exhibited a greater overall loss in performance compared to SNR-based pruning for MP, both approaches began to experience performance degradation at 95% of the network being pruned, as reflected in the CDFs for variance and SNR. Additionally, in the BBB network, variance-based pruning performed significantly worse than SNR-based pruning, indicating that variance may not be as reliable as a parameter importance measure in the BBB framework.

Based on the pruning performance of the MP framework, trained MP networks can accurately discern important parameters via learned parameter uncertainty. This inherent ability for self-determination of relevant parameters is a crucial tool for CL. Important parameters can be appropriately regularized to avoid changes and prevent catastrophic interference, ensuring the retention of knowledge from previous tasks. In contrast, unimportant parameters, which have demonstrated minimal contribution to network predictions, can be used for learning subsequent information, making MP beneficial in Continual Learning scenarios.

IV. Bayesian Continual Learning

Continual Learning (CL) can be divided into three subcategories: *Task Incremental Learning*, *Domain Incremental Learning*, and *Class Incremental Learning*, where the premise of learning and retaining information over multiple training periods remains the same, but how context is provided changes in each scenario [23]. Methods described in this paper will focus on *Task Incremental Learning*, in which the context $\mathcal{C}$ for the input $\mathcal{X}$ changes over time and is provided during training and inference periods. The output space $\boldsymbol{y}$ is separated for

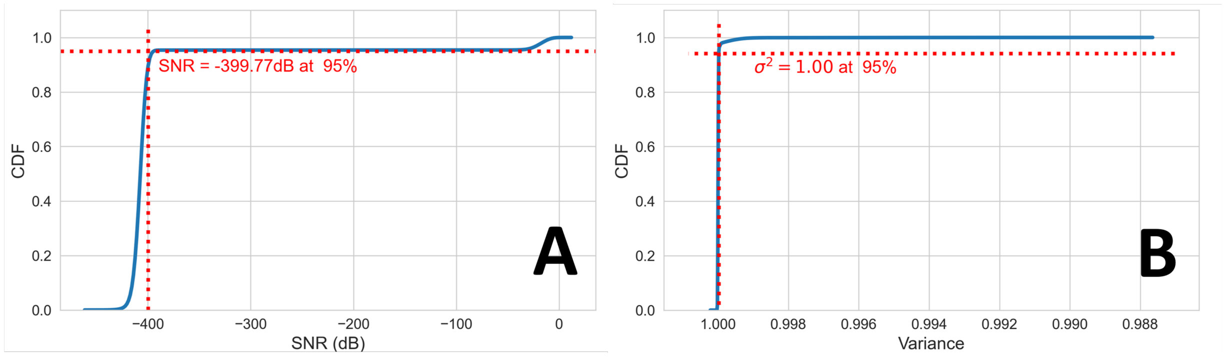

Fig. 1. Analysis of parameter uncertainty from a two 800-node hidden layer fully connected network (A) Cumulative Distribution Function plot of the Signal-to-Noise Ratio (SNR) demonstrating 95% of the parameters are approximately -400dB SNR. (B) Cumulative Distribution Function plot of the Variance demonstrating 95% of the parameters have a variance of 1 or greater.

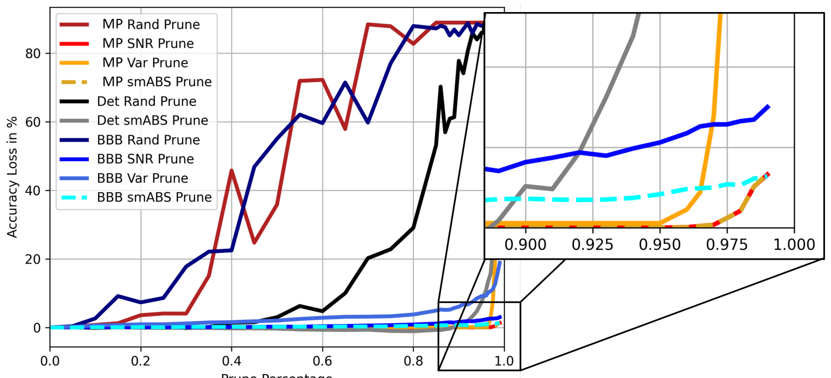

Fig. 2. Loss in performance from the original validation accuracy as a result of various pruning methods. Moment Propagation performance is presented in warm colors, Bayes-by-Backprop (BBB) performance in cool colors, and deterministic performance in grey and black.

each input context, producing the mapping $\boldsymbol{f} : \mathcal{X} \times \mathcal{C} \rightarrow \boldsymbol{y}$ [23]. This additional task context is only leveraged for the multi-headed network architecture in which task information is required to choose the appropriate output head and is not used to select which network parameters are active.

A. Learning Rate Adaptation

Initially explored by Ebrahimi et al. [24], *Learning Rate Adaptation* (LRA) leverages learned parameter uncertainty to adapt the learning rates of individual parameters according to their relevance to the network's functional approximation. Lower learning rates for important network parameters ideally prevents catastrophic interference in these parameters by restricting change while allowing unimportant parameters to learn new information freely. Our approach to Learning Rate Adaption replaces the Bayes-by-Backprop (BBB) framework with the Moment Propagation framework. Instead of using a BBB parameter's latent sampling distribution to determine parameter importance, parameter importance is determined by a random variable that directly contributes to the network's predictive distribution.

Learning rates for each parameter are determined by mapping parameter importance from all network parameters to a user-defined range of learning rates. First, parameter importance is determined according to the parameter's variance, $1/\sigma^2_{\omega_i}$, or SNR, $|\mu_{\omega_i}|/\sigma^2_{\omega_i}$. The resulting importance values are remapped across the whole network, excluding the classification head, instead of across each layer given there is no guarantee that parameter importance will be even distributed across all layers. The remapping function is shown in Equation (7), where $\boldsymbol{\alpha}_{t+1}$ represents the vector of all parameter learning rates for the next task, $\boldsymbol{\iota}_t$ represents the vector of all parameter importance values from task, t. The min and max user-defined range of learning rate values are defined as α_{min} and α_{min}. Learning rates for each parameter's mean and variance are updated synchronously because the approximating distribution for each parameter is treated as a single parameter. Ultimately, the mean and variance of the most important parameters will receive the lowest user-defined learning rate, restricting any change in the parameter. The mean and variance of the most uncertain parameters will receive the highest, allowing these parameters to learn freely.

$$\boldsymbol{\alpha}_{t+1} \leftarrow \frac{((\boldsymbol{\iota}_t - \min(\boldsymbol{\iota}_t))(\alpha_{\text{max}} - \alpha_{\text{min}})}{\max(\boldsymbol{\iota}_t) - \min(\boldsymbol{\iota}_t)} + \alpha_{\text{min}} \tag{7}$$

B. Per-Parameter Bayesian Inference

Our second approach, *Per-Parameter Bayesian Inference* (PPBI), leverages the same concept of leveraging parameter uncertainty but instead changes the weighting of the KL regularization term within the ELBO. Drawing inspiration from Ebrahimi et al. [24], Ahn et al. [25], and Nguyen et al. [26], this framework performs approximate Bayesian Inference on a per-parameter basis guided by parameter uncertainty without any changes to the regularization term. PPBI applies a similar methodology to Learning Rate Adaption; however, for each new task, a KL regularization term weighting value is applied according to each parameter's importance. Before training a task the network prior is replaced with the previous tasks learned posterior. To control the explicit regularization of every parameter, regularization weights are adjusted based on each parameter's importance by mapping parameter importance values to a user-defined range of regularization weighting values. For this technique, the most important parameters will map to the highest user-defined weighting, heavily restricting change from the previous tasks posterior (the prior), while the least important parameters will map to the lowest user-defined weighting, allowing those parameters to change easily to maximize the network log-likelihood for the current task. For PPBI, the remapping function is shown in Equation (8).

$$\boldsymbol{\tau}_{t+1} \leftarrow \frac{((\boldsymbol{\iota}_t - \min(\boldsymbol{\iota}_t))(\tau_{\text{min}} - \tau_{\text{max}})}{\max(\boldsymbol{\iota}_t) - \min(\boldsymbol{\iota}_t)} + \tau_{\text{max}} \tag{8}$$

The algorithm for MP-based CL for both LRA and PPBI is presented in Algorithm 1.

C. Experimental Setup

1) Datasets and Networks: Our CL methodologies are evaluated for eight different CL benchmark datasets of increasing difficulty: Two Split MNIST, Five Split MNIST, Permuted

Algorithm 1 Moment Propagation Continual Learning

```
Require: ${\mathcal{D}_t}_{t\in\mathcal{T}}$;  $\mathcal{D}_t = (\mathcal{X}_t, \boldsymbol{y}_t)$
         $q_{\boldsymbol{\theta}_0}(\boldsymbol{\Omega_0}) \sim \prod_{j=1}^{|\boldsymbol{\Omega}_0|} \mathcal{N}(\mu_{w_0^{(j)}}, \sigma^2_{w_0^{(j)}})$
         $p_0(\boldsymbol{\Omega}_0) \sim \prod_{j=1}^{|\boldsymbol{\Omega}_0|} \mathcal{N}(0,1)$
         $\boldsymbol{\tau}_0 = (\tau_0^{(j)} \cdots \tau_0^{(j)})$
         $\boldsymbol{\alpha}_0 = (\alpha_0^{(j)} \cdots \alpha_0^{(j)})$
1:  for all tasks, $t$, do
2:    repeat
3:      for all examples in $\mathcal{D}_t$ do
4:        $\mathcal{L}_{\text{LL}} \leftarrow \mathbb{E}_{q_{\boldsymbol{\theta}_t}(\boldsymbol{\Omega}_t)}[\ln p(\hat{\boldsymbol{y}}|\mathcal{X}_t, \boldsymbol{\Omega}_t)]$
5:        $\mathcal{L}_{\text{KL}} \leftarrow \sum_{j=1}^{|\boldsymbol{\Omega}_t|} \tau_t^{(j)} \text{KL}_{q_{\boldsymbol{\theta}_t}(\boldsymbol{\Omega}_t^{(j)})}[q_{\boldsymbol{\theta}_t}(\boldsymbol{\Omega}_t^{(j)})||p_t(\boldsymbol{\Omega}_t^{(j)})]$
6:        $\mathcal{L}_{\text{VDP}} \leftarrow \mathcal{L}_{\text{LL}} - \mathcal{L}_{\text{KL}}$
7:        for each network parameter, $\boldsymbol{\Omega}_t^{(j)}$, do
8:          $\boldsymbol{\Omega}_t^{(j)} \leftarrow \boldsymbol{\Omega}_t^{(j)} - \boldsymbol{\alpha}_t^{(j)} \frac{\partial \mathcal{L}_{\text{VDP}}}{\partial \boldsymbol{\Omega}_t^{(j)}}$
9:        end for
10:     end for
11:   until validation accuracy plateaus
12:   for each network parameter, $\boldsymbol{\Omega}_t^{(j)}$, do
13:     $\iota_t^{(j)} \leftarrow 1/\sigma^2_{w_j}$ or $\mu_{w_j}/\sigma^2_{w_j}$
14:   end for
15:   if Learning Rate Adaptation then
16:     $\boldsymbol{\alpha}_{t+1} \leftarrow \frac{((\boldsymbol{\iota}_t - \min(\boldsymbol{\iota}_t))(\alpha_{\max} - \alpha_{\min})}{\max(\boldsymbol{\iota}_t) - \min(\boldsymbol{\iota}_t)} + \alpha_{\min}$
17:     $\boldsymbol{\tau}_{t+1} \leftarrow \boldsymbol{\tau}_t$
18:   end if
19:   if Per-Parameter Bayesian Inference then
20:     $\boldsymbol{\tau}_{t+1} \leftarrow \frac{((\boldsymbol{\iota}_t - \min(\boldsymbol{\iota}_t))(\tau_{\min} - \tau_{\max})}{\max(\boldsymbol{\iota}_t) - \min(\boldsymbol{\iota}_t)} + \tau_{\max}$
21:     $p_{t+1}(\boldsymbol{\Omega}_{t+1}) \leftarrow q_{\boldsymbol{\theta}_t}(\boldsymbol{\Omega}_t)$
22:     $\boldsymbol{\alpha}_{t+1} \leftarrow \boldsymbol{\alpha}_t$
23:   end if
24: end for
```

MNIST, Two Split CIFAR10, Five Split CIFAR10, Mixed CIFAR10-CIFAR100, and a sequence of eight datasets. Split MNIST and CIFAR10 datasets consist of each base dataset separated into two tasks of five classes and five tasks of 2 classes, for two split and five split, respectively. The Permuted MNIST dataset consists of ten different pixel level permutations applied to the entire base MNIST dataset, resulting in ten tasks of ten classes each. Mixed CIFAR10-CIFAR100 combines the base CIFAR10 and CIFAR100 benchmark datasets and alternates between tasks of two classes from CIFAR10 and twenty classes from CIFAR100, for ten tasks. Finally, a sequence of eight datasets is evaluated, consisting of MNIST, CIFAR10, CIFAR100, NotMNIST, SVHN, Traffic Signs, FaceScrub, and FashionMNIST where each task is a new dataset.

For all sequential approaches, 15% of the training set for each task is reserved for validation, while the test set is exclusively used for testing after task training is complete. The order of each dataset is randomized for each epoch, including for joint training. All datasets are normalized to the mean and the standard deviation of the full dataset before splitting. For split CIFAR10 and mixed CIFAR10/CIFAR100, a random crop with a padding of 4 pixels and a random horizontal flip are added to assist with generalization. These transforms are added to existing approaches for a fair comparison. Transform was not applied to the sequence of eight datasets, as it adversely affected performance, but datasets are padded to a image size of 32x32 pixel, and input data consisting of one channel are replicated across two additional channels for a consistent input image size of 32x32x3.

A two 800-node hidden layer fully connected network architecture is used across all approaches to compare continual learning performance for two-split, five-split, and permuted MNIST datasets. An AlexNet Convolutional Neural Network architecture is used for two-split and five-split CIFAR10, mixed CIFAR10-CIFAR100, and the sequence of eight datasets. We recollect all results for these architectures and datasets, slightly improving on some previously reported results due to differences in architectures used.

2) Hyperparameters: A grid search is performed for hyperparameters to maximize the performance across all tasks. Given that the variance of each parameter directly contributes to the predictive distribution, the initialization of these values has a significant impact on overall performance. Thus, parameter variance initialization is searched between $\sigma^2_\pi \leftarrow [-10, -18]$.

Regularization toward the prior also has a significant impact on performance. The initial KL weighting is searched between $\tau_0 \leftarrow [1e\text{-}3, 1e\text{-}8]$ and is initially applied to all parameters regardless of approach. Values higher than $1e$-3 typically cause too much regularization toward the prior, resulting in poor performance due to the network's inability to learn a sufficiently complex representation for the task. For LRA, the maximum learning rate for the mapping to the user-defined range is searched between $\alpha_{\max} \leftarrow [1e\text{-}3, 1e\text{-}5]$. For KL Weight Adaptation, the maximum KL weighting for the mapping to the user-defined range is searched between $\tau_{\max} \leftarrow [1e\text{-}2, 1e\text{-}7]$. The minimum learning rate and KL weighting are both set to $1e$-12 and are not tuned. The number of epochs and batch size were fixed at 250 and 500, respectively. A large batch size was chosen because more performance was retained in deterministic baseline tests with Fine Tuning and enabled more efficient use of computational resources.

D. Performance Measurement

Performance is gauged through the Average Test Classification Accuracy (ACC) and Backward Transfer (BWT). Average Test Classification Accuracy is an average of all test accuracies on individual tasks after training all tasks. Backward Transfer indicates how much learning new information has affected performance on previous tasks. Backward Transfer values less than zero indicate catastrophic forgetting, while values greater than zero indicate improved performance on previous tasks after training on new information [24]. These metrics are shown in Equation 14.

$$\mathrm{BWT} = \frac{1}{t}\sum_{i=1}^{t} R_{i,t} - R_{i,i}$$
$$\mathrm{ACC} = \frac{1}{t}\sum_{i=1}^{t} R_{i,t} \qquad (14)$$

V. Results and Discussion

Our CL methodologies are compared to parameter uncertainty-based methods: Uncertainty-Based Continual Learning (UCL) [25] and Uncertainty-Guided Continual Learning (UCB) [24]. We also compare our methods to long-standing standards for continual learning: Elastic Weight Consolidation (EWC) [2], Memory Aware Synapses (MAS) [7], Synaptic Intelligence (SI) [6], and Hard Attention to Task (HAT) [8]. These approaches to catastrophic interference mitigation are compared to baselines using Moment Propagation Framework: Fine Tuning (FT), where no efforts are made to mitigate catastrophic interference, Feature Freezing (FF), where all but the classification head is frozen after training the first task, and Joint Training (JT), where tasks are trained sequentially, but jointly with previous tasks. Fine Tuning represents the lower bound on performance, indicating performance was not gained over sequential training, while Joint Training represents the theoretical upper bound on performance. The performance results for all listed methods are presented in Table I, where the maximum ACC and maximum BWT values for each dataset are shown in bold.

Our MP-based LRA and PPBI methodologies outperform their sampling-based predecessors, UCB and UCL, respectively. This performance improvement can be attributed to better measures of parameter uncertainty resulting from the MP framework. In UCB, performance improved as the number of samples of the predictive distribution increases [24]. Approximating the predictive distribution via the propagated moments improves measures of the network log-likelihood. The directly differentiable network log-likelihood then improves gradient updates of the network parameter, improving the measure of parameter uncertainty and, thus, parameter importance. Similarly, UCL only leverages one sample of the predictive distribution to estimate the network likelihood [25] but outperforms UCB on more complex benchmark datasets by heavily relying on changes made to the KL regularization term in the ELBO to further restrict parameter change. Additionally, our methods perform on par with HAT's Average Test Accuracy despite not having a propagation mask to select which features are used for which tasks. This masks helps HAT maintain a BWT of near zero by not only freeze important parameters to restrict parameter change, but also masking out all other unimportant parameters to prevent interference on a feature level.

Despite improved performance over previous approaches, results for LRA and PPBI only show a marginal performance improvement over hyperparameter tuning the FF method. This implies that freezing a MP DNN after training the first task and learning a new classification layer can provide reasonable Task Incremental Learning performance. This trend differed slightly for the mixed CIFAR10 CIFAR100 dataset, where the ACC for FF was lower than uncertainty-based methods, indicating the network did not gain enough information from the training of the first task, two classes of CIFAR10, to sufficiently adapt to CIFAR100 based tasks with only learning the classification layer.

A. Split and Permuted MNIST

PPBI slightly outperforms LRA for split and permuted MNIST benchmark sets. For two-split MNIST, PPBI achieves an Average Test Accuracy of 99.40% and a Backward Transfer of -0.05% with a Variance-based uncertainty metric. For five-split MNIST, PPBI achieves an Average Test Accuracy of 99.84% and a Backward Transfer of -0.04% with an SNR-based uncertainty metric. For permuted MNIST with ten tasks, PPBI achieves an Average Test Accuracy of 98.19% and a Backward Transfer of -0.25% with a variance-based uncertainty metric. The restrictive nature of Bayesian Inference minimizing the KL divergence between the variational posterior and prior may have helped in this scenario, given that all tasks are distributed. Thus, restricting change from one

TABLE I
Task Incremental Learning Results

Approach	**2-Split MNIST**		**5-Split MNIST**		**Permuted MNIST**		**2-Split CIFAR10**		**5-Split CIFAR10**		**CIFAR10/100**		**Sequence**	
	ACC	**BWT**	**ACC**	**BWT**	**ACC**	**BWT**	**ACC**	**BWT**	**ACC**	**BWT**	**ACC**	**BWT**	**ACC**	**BWT**
EWC	98.12%	-1.03%	99.18%	-0.57%	96.19%	-0.46%	82.78%	-0.48%	86.01%	-3.17%	69.28%	-4.20%	70.81%	-3.05%
SI	98.85%	-0.08%	99.47%	-0.24%	93.88%	-0.06%	86.19%	-1.49%	86.85%	-5.74%	70.39%	-9.04%	61.91%	-16.04%
MAS	98.82%	-0.13%	99.57%	-0.01%	95.46%	-0.21%	99.46%	-2.28%	85.98%	-7.00%	70.70%	-8.56%	60.62%	-13.84%
UCL	98.66%	-0.41%	99.21%	-0.56%	94.97%	-2.58%	82.02%	-2.46%	84.26%	-6.78%	71.68%	-4.16%	69.81%	-2.10%
UCB	99.18%	**+0.01%**	99.63%	0.00%	97.42%	**0.00%**	73.76%	-5.10%	76.67%	-7.56%	60.64%	-15.02%	48.37%	-24.90%
LRA (Var.)	99.32%	-0.04%	99.85%	**+0.02%**	97.40%	-0.81%	**90.83%**	**-0.30%**	**92.17%**	-0.28%	**77.16%**	-1.43%	**77.40%**	-0.64%
LRA (SNR)	99.29%	-0.02%	99.80%	-0.03%	97.65%	-0.44%	89.61%	-0.96%	91.69%	-0.91%	76.58%	-4.22%	76.94%	-1.15%
PPBI (Var.)	**99.40%**	-0.05%	99.81%	-0.06%	**98.19%**	-0.25%	89.29%	-0.60%	89.78%	**+0.04%**	75.12%	-1.31%	77.04%	**-0.10%**
PPBI (SNR)	99.37%	-0.06%	**99.84%**	-0.04%	98.18%	-0.27%	89.46%	-1.02%	89.99%	-0.06%	75.02%	**-0.29%**	77.15%	-0.61%
HAT	98.79%	0.00%	99.75%	0.00%	97.34%	0.00%	86.91%	0.01%	92.09%	0.00%	77.84%	-0.03%	83.13%	-0.03%
MP-FT	98.98%	-0.20%	98.32%	-1.46%	95.91%	-2.47%	85.72%	-6.30%	79.41%	-15.46%	57.80%	-19.55%	62.98%	-20.44%
MP-FF	98.91%	0.00%	99.40%	0.00%	96.72%	0.00%	88.72%	0.00%	89.54%	0.00%	69.27%	-0.25%	76.70%	-0.01%
MP-JT	99.39%	0.00%	99.73%	-0.08%	98.15%	-0.17%	91.51%	+1.14%	94.64%	+0.79%	87.46%	+1.23%	78.27%	-1.08%

task to the next may have had less impact than more advanced tasks.

B. Split CIFAR10 and Mixed CIFAR10/100

Conversely, for Split CIFAR10 and Mixed CIFAR10/CIFAR100, LRA sightly outperforms PPBI. For two-split CIFAR10, LRA achieves an Average Test Accuracy of 90.83% and a Backward Transfer of -0.30% with a Variance-based uncertainty metric. For five-split CIFAR10, LRA achieves an Average Test Accuracy of 92.17% and a Backward Transfer of -0.28% with a Variance-based uncertainty metric. For mixed CIFAR10 CIFAR100, LRA achieves an Average Test Accuracy of 77.16% and a Backward Transfer of -0.29% with a Variance-based uncertainty metric. We attribute these results to the same characteristics of Per-parameter Bayesian Inference, which benefited Split and Permuted MNIST tasks. However, in the case of CIFAR10 and CIFAR100 datasets separating classes into tasks may require parameters to diverge from the previous tasks posterior. Thus, PPBI may cause unwanted regularization to the network prior. Learning Rate Adaptation avoids this problem by restricting all changes in important parameters while allowing unimportant parameters to move away from the network posterior to the previous task.

C. Sequence of Eight Datasets

For the Sequence of Eight Datasets PPBI and LRA, perform similarly. LRA provides the maximum Average Test Accuracy at 77.40% with a Backward Transfer of -0.64% with a variance-based uncertainty metric. PPBI and LRA significantly outperform all other methods apart from HAT.

VI. Conclusion

In this work, two Continual Learning methodologies are presented that leverage learned parameter uncertainty derived from a Moment Propagation framework to regularize training of new tasks to prevent Catastrophic Forgetting. These methods are detailed by deriving the custom layers for a basic Convolutional Neural Network from the principles of Variational Inference. The concept of parameter importance through learned uncertainty from the Moment Propagation framework's is demonstrated and applied to Continual Learning through two methodologies to Learning Rate Adaptation and Per-parameter Bayesian Inference. While these approaches leverage learned parameter importance in different ways, both mitigate catastrophic forgetting through regulariz the learning of network parameters. These methods were evaluated on multiple sequential benchmark datasets, and performance was compared to other similar previously published approaches. Ultimately, Learning Rate Adaptation and Per-parameter Bayesian Inference outperform or yield comparable results to existing approaches through improved measures of parameter uncertainty.

References

[1] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," vol. 10, no. 4, pp. 12–25. Conference Name: IEEE Computational Intelligence Magazine.

[2] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," vol. 114, no. 13, pp. 3521–3526.

[3] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," vol. 113, pp. 54–71.

[4] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of Learning and Motivation* (G. H. Bower, ed.), vol. 24, pp. 109–165, Academic Press.

[5] R. M. French, "Catastrophic forgetting in connectionist networks," vol. 3, no. 4, pp. 128–135.

[6] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence."

[7] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget."

[8] J. Serrà, D. Surís, M. Miron, and A. Karatzoglou, "Overcoming catastrophic forgetting with hard attention to the task."

[9] J. Yoon, E. Yang, J. Lee, and S. J. Hwang, "Lifelong Learning with Dynamically Expandable Networks," June 2018. arXiv:1708.01547 [cs].

[10] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive Neural Networks," Oct. 2022. arXiv:1606.04671 [cs].

[11] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental Classifier and Representation Learning," Apr. 2017. arXiv:1611.07725 [cs, stat].

[12] J. Hurtado, A. Raymond-Saez, V. Araujo, V. Lomonaco, A. Soto, and D. Bacciu, "Memory Population in Continual Learning via Outlier Elimination," Oct. 2023. arXiv:2207.01145 [cs].

[13] E. Belouadah and A. Popescu, "IL2M: Class Incremental Learning With Dual Memory," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 583–592, Oct. 2019. ISSN: 2380-7504.

[14] R. Kemker and C. Kanan, "FearNet: Brain-Inspired Model for Incremental Learning," Feb. 2018. arXiv:1711.10563 [cs].

[15] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient Lifelong Learning with A-GEM," Jan. 2019. arXiv:1812.00420 [cs, stat].

[16] D. Lopez-Paz and M. Ranzato, "Gradient Episodic Memory for Continual Learning," Sept. 2022. arXiv:1706.08840 [cs].

[17] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual Learning with Deep Generative Replay," Dec. 2017. arXiv:1705.08690 [cs].

[18] D. Dera, N. C. Bouaynaya, G. Rasool, R. Shterenberg, and H. M. Fathallah-Shaykh, "PremiUm-CNN: Propagating uncertainty towards robust convolutional neural networks," vol. 69, pp. 4669–4684. Conference Name: IEEE Transactions on Signal Processing.

[19] D. P. Kingma and M. Welling, "Auto-encoding variational bayes."

[20] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network," in *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1613–1622, PMLR. ISSN: 1938-7228.

[21] G. E. Hinton and D. van Camp, "Keeping the neural networks simple by minimizing the description length of the weights," in *Proceedings of the sixth annual conference on Computational learning theory*, COLT '93, pp. 5–13, Association for Computing Machinery.

[22] X. He and M. Lin, "Continual Learning from the Perspective of Compression," *arXiv e-prints*, p. arXiv:2006.15078, June 2020. doi: 10.48550/arXiv.2006.15078.

[23] G. M. van de Ven, T. Tuytelaars, and A. S. Tolias, "Three types of incremental learning," vol. 4, no. 12, pp. 1185–1197. Number: 12 Publisher: Nature Publishing Group.

[24] S. Ebrahimi, M. Elhoseiny, T. Darrell, and M. Rohrbach, "Uncertainty-guided continual learning with bayesian neural networks."

[25] H. Ahn, S. Cha, D. Lee, and T. Moon, "Uncertainty-based continual learning with adaptive regularization," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc.

[26] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner, "Variational continual learning."