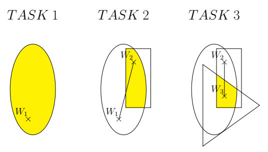
Continual Learning with Guarantees via Weight Interval Constraints

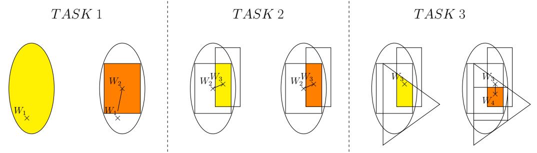
Maciej Wołczyk [*1] Karol J. Piczak [*1] Bartosz Wójcik [1] Łukasz Pustelnik [1] Paweł Morawiecki [2] Jacek Tabor [1] Tomasz Trzciński [1 3 4] Przemysław Spurek [1]

Abstract

We introduce a new training paradigm that enforces interval constraints on neural network parameter space to control forgetting. Contemporary Continual Learning (CL) methods focus on training neural networks efficiently from a stream of data, while reducing the negative impact of catastrophic forgetting, yet they do not provide any firm guarantees that network performance will not deteriorate uncontrollably over time. In this work, we show how to put bounds on forgetting by reformulating continual learning of a model as a continual contraction of its parameter space. To that end, we propose Hyperrectangle Training, a new training methodology where each task is represented by a hyperrectangle in the parameter space, fully contained in the hyperrectangles of the previous tasks. This formulation reduces the NP-hard CL problem back to polynomial time while providing full resilience against forgetting. We validate our claim by developing InterContiNet (Interval Continual Learning) algorithm which leverages interval arithmetic to effectively model parameter regions as hyperrectangles. Through experimental results, we show that our approach performs well in a continual learning setup without storing data from previous tasks.



(a) General CL setting.



(b) InterContiNet CL setting.

Figure 1. Visualization of parameter space regions that perform well on tasks learned sequentially. For a new task, weights θ_i must lie at the intersection of consecutive regions for the model to perform well on previous tasks. In the general CL setting, shown in Figure 1(a) from (Knoblauch et al., 2020), such areas can have arbitrary shapes. We introduce the Hyperrectangle Training paradigm where the parameter spaces are modeled with hyperrectangles, colored orange in Figure 1(b). This formulation puts guarantees on forgetting while reducing the calculation of NP-hard CL problem back to polynomial time.

1. Introduction

Learning from a continuous stream of data is natural for humans, as new experiences come sequentially in our life. Yet, artificial neural network models fail to exhibit the very same skill (McCloskey & Cohen, 1989; Ratcliff, 1990; French, 1999; Goodfellow et al., 2014). Although they generally deal well with solving increasingly complex tasks, their inability to acquire new knowledge without *catastrophically forgetting* what they learned previously is considered one of the critical roadblocks to reaching human-like intelligence.

Continual learning is a rapidly growing field of machine learning that aims to solve this limitation and to bridge the gap between human and machine intelligence. Although several methods effective at reducing forgetting when learning new tasks were proposed (Kirkpatrick et al., 2017; Lee et al., 2017; Li & Hoiem, 2017; Lopez-Paz & Ranzato, 2017; Shin et al., 2017; Zenke et al., 2017; Aljundi et al., 2018; Masse et al., 2018; Liu et al., 2018; Rolnick et al., 2019; van de Ven et al., 2020), they typically do not provide any firm guarantees about the degree of forgetting experienced by the model, which renders them inapt for safety-critical applications, such as autonomous driving or robotic manipulation. For instance, an autonomous car that was trained online during winter cannot forget how to drive in the snow, after acquiring new driving skills during summer. On the

[*]Equal contribution [1]Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland [2]Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland [3]Warsaw University of Technology, Warsaw, Poland [4]Tooploox. Correspondence to: Maciej Wołczyk <maciej.wolczyk@doctoral.uj.edu.pl>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

other hand, methods that do guarantee lack of forgetting, e.g. Rusu et al. (2016); Mallya & Lazebnik (2018); Mallya et al. (2018), operate in conditions that are far from real life — they assume that the task identity is known at inference, and hence we are allowed to adapt the network for each task.

In this work, we identify those limitations of prior research and introduce a new learning paradigm for continual learning which puts strict bounds on forgetting, while being suitable for a more realistic single-head scenario. More specifically, we formulate continual learning as a constrained optimization problem, where when learning a new task T the optimization needs to remain in the parameter region that prevents performance drop on all previous tasks $1, \ldots, T-1$. In general, this problem is NP-hard (Knoblauch et al., 2020) because the shapes of the viable parameter regions are highly irregular and overlaps are hard to identify. To overcome this problem, we propose to look for the intersections of hyperrectangles which are subsets of these regions, as displayed in Figure 1. We show that finding those intersections can be done in polynomial time, as long as we trim the parameter regions to fit the consecutive hyperrectangles. The resulting training paradigm, dubbed Hyperrectangle Training, is able to learn from a sequence of tasks while putting bounds on catastrophic forgetting.

Although this simple idea is theoretically viable, finding hyperrectangles that follow these requirements and performing optimization with constrained parameter space is not trivial with the existing neural network models. To solve this problem, we propose InterContiNet, a novel neural algorithm based on interval arithmetic which we use to implement the Hyperrectangle Training. Instead of optimizing individual network parameters, we consider intervals of parameter values, ensuring that every parameter value within the interval does not lead to performance deterioration with respect to the previous tasks. More specifically, once training on task T is finished, we use the obtained intervals as the hyperrectangle constraints for training the following tasks $(T+1, T+2, \ldots)$. We show how InterContiNet can be incorporated within conventional neural network building blocks such as linear layers, convolutional layers, and ReLU activations. The resulting combination of Hyperrectangle Training and InterContiNet algorithm offers competitive results on a range of continual learning tasks we evaluated.

To summarize, our main contributions are the following:

- A novel paradigm of continual learning called Hyperrectangle Training that models optimal parameter space with hyperrectangles, allowing us to control catastrophic forgetting in neural networks and set guaranteed performance. We show that with adequate parameterization we can use this paradigm to train neural networks in polynomial time.
- InterContiNet, a new algorithm that implements Hyperrectangle Training in practice, leveraging interval arithmetic of model parameters.
- Thorough experimental evaluation that confirms the validity of our method and its potential for putting guarantees on catastrophic forgetting in contemporary deep networks.

2. Related Works

Continual learning Traditionally, continual learning approaches are grouped into three general families: regularization, dynamic architectures, and replay (Parisi et al., 2019; Delange et al., 2021).

Regularization-based methods extend the loss function with additional terms slowing down changes in model parameters crucial for previous tasks. These terms can be introduced individually for each task or as a single overall value representative for the whole sequence (*online* variant). Based on the type of regularization, a further subdivision splits these methods into prior- and data-focused approaches.

Prior-focused methods employ a prior on the model parameters when learning new tasks. The importance of individual parameters is estimated, i.a., with the Fisher information matrix, as shown in the seminal work on Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017). A similar approach, Synaptic Intelligence (Zenke et al., 2017), utilizes the whole learning trajectory to compute this importance measure. In contrast, the Memory Aware Synapses technique (MAS) (Aljundi et al., 2018) approximates the importance of the parameters based on the gradients of the squared L_2-norm of the learned function output. Our proposed method bears a certain resemblance to these methods, as it also regularizes the model's parameters. However, we approach this problem differently by abandoning the soft quadratic penalties and instead introducing hard constraints based on the interval propagation loss. This allows us to provide guarantees on the worst-case level of forgetting.

Data-focused methods instead perform knowledge distillation from models trained on previous tasks when learning on new data. For instance, the Learning without Forgetting paradigm (LwF) (Li & Hoiem, 2017) employs an additional distillation loss based on the comparison of new task outputs generated by the new and old models.

Regularization-based methods have a number of advantages. They work without changing the structure of the model and without storing any examples of the old data, which might be crucial due to data privacy issues. However, as they only impose a soft penalty, both the prior- and data-focused regularization methods cannot entirely prevent the forgetting of previous knowledge.

Such guarantees can be provided instead by approaches with dynamic architectures, which dedicate separate model branches to different tasks. These branches can be grown progressively, such as in the case of Progressive Neural Networks (Rusu et al., 2016). Alternatively, a static architecture can be reused with iterative pruning proposed by PackNet (Mallya & Lazebnik, 2018) or by using Supermasks in Superposition (Wortsman et al., 2020). A major practical drawback of these methods is that they require the knowledge of the actual task identity during inference, which is problematic in more realistic scenarios. Therefore, due to this limitation, we do not consider them further in our analysis.

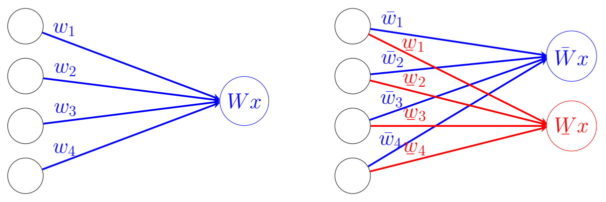
A different, very successful approach in continual learning is replaying some form of old data during incremental training to maintain the knowledge acquired in the past (Lopez-Paz & Ranzato, 2017; Shin et al., 2017; Rolnick et al., 2019). This process can use actual raw examples from the previous tasks or samples generated synthetically by a separate model. However, this paradigm is inapplicable in various applications, such as working with private medical data. Thus, in further experiments, we limit our comparisons to approaches based on regularization.

Finally, a recent work (Mirzadeh et al., 2021) addressed catastrophic forgetting by exploiting the linear connectivity between solutions obtained through multitask and continual learning in the form of Mode Connectivity SGD. Our work shares its fundamental motivation, i.e. driving the optimization process to parameter subspaces appropriate for the whole sequence of tasks. However, in contrast to Mirzadeh et al. (2021), we do not employ any kind of replay buffer, but we establish the solution boundaries based on interval arithmetic.

Interval arithmetic for neural networks In deep learning, interval arithmetic was used in three main areas. First of all, we can use interval arithmetic to deal with the situation where we have only uncertain information about the data. Chakraverty & Sahoo (2014) presented an interval artificial neural network (IANN) that can handle input and output data represented as intervals. This architecture was further adapted for viruses data with uncertainty modeled by intervals (Chakraverty & Sahoo, 2017; Sahoo & Chakraverty, 2020).

In Gowal et al. (2018); Morawiecki et al. (2019) interval arithmetic was used to produce neural networks that are robust against adversarial attacks. In such an approach authors used worst-case cross-entropy to train the classification model. In Proszewska et al. (2021), the architecture was employed for representing voxels in 3D shape modeling.

In contrast to these previous works, we apply interval arithmetic to weights instead of input data.



(a) The classical neuron uses weight W. As an output of a single classical neuron, we obtain a vector of activations Wx.

(b) The interval neuron uses two weight matrices $\underline{W}$ and $\bar{W}$. As an output of a single interval neuron we obtain an interval $\underline{W}x, \bar{W}x$.

Figure 2. Comparison between the classical neuron and the interval neuron. In the interval neuron, we have two sets of weights describing the intervals.

3. Hyperrectangle Training

In this section, we will present Hyperrectangle Training for the continual learning problem. The main idea is to constrain the parameter search within the set of parameters for which any particular solution is valid for the previous tasks. In turn, we can guarantee that any parameter obtained this way will still be a valid solution for the previous task, thus putting bounds on forgetting. Although explicitly representing the region of valid solutions is not tractable, we show that using interval arithmetic we can efficiently compute the upper bound of the worst-case loss in this region to facilitate practical implementations. This general approach is presented in Figure 1.

The continual learning problem Assuming a sequence of tasks T_i for $i = 1, \ldots, k$ we formulate the continual learning objective for task T_j following Chaudhry et al. (2019) as:

$$\arg\min_{\theta} \ell(T_j, \theta) \text{ satisfying } \ell(T_m, \theta) \leq \ell(T_m, \theta^*_m), \quad (1)$$

for all $m = 1, \ldots, j-1$, where θ^*_m are the parameters obtained directly after learning task m and ℓ is the cross-entropy loss over the whole task dataset $\ell(T_j, \theta) = \frac{1}{|T_j|}\sum_{(x,y)\in T_j} l(\phi_\theta(x), y)$, where by $\phi_\theta(x)$ we denote the (pre-softmax) logits produced by the neural network ϕ with parameters θ. In other words, we would like to train our model on j-th task without reducing the performance of previous tasks $m = 1, \ldots, j-1$.

Since this constraint is difficult to satisfy given the limited access to data from previous tasks, most continual learning methods transform it into a soft penalty. In particular, regularization-based continual learning methods such as EWC (Kirkpatrick et al., 2017) usually use a soft approximation of the forgetting constraint. While this approach reduces changes in weights which are considered important for the previous task, it does not completely eliminate forgetting. Rephrasing this in the context of stability-plasticity trade-off, most methods allow for better plasticity at the cost of sometimes sacrificing stability. However, in many applications, such as medical imaging, autonomous vehicles, or robotics, even slight hints of forgetting are not acceptable. Our main goal is to create a CL method that does not allow any forgetting while maintaining good plasticity.

Hyperrectangle Training In order to fully realize the constraint in Eq. (1) we change the approach of thinking about the parameter space of the neural networks, switching the focus from finding particular points $\theta \in \mathbb{R}^D$ in the parameter space to reasoning about whole regions $\Theta \subset \mathbb{R}^D$. In other words, in the continual learning problem of optimizing over $T_1, \dots T_n$, when training on task T_j, we are interested in finding:

$$\begin{aligned}\Theta_{j+1} = \arg\min_{\Theta} \max_{\theta \in \Theta_j} \ell(T_{j+1}, \theta) \\ \text{satisfying } \|\Theta\| > C_{j+1}, \Theta \subseteq \Theta_j\end{aligned} \tag{2}$$

where $\|\Theta\|$ denotes some notion of the size of the set, e.g. its volume, C_{j+1} is a hyperparameter denoting the minimal required size of that set, and we set $\Theta_0 = \mathbb{R}^D$ to be the whole domain of real vectors. With this formulation, since $\Theta_{j+1} \subseteq \Theta_j$ we can easily see that for any $\theta \in \Theta_{j+1}, \ell(T_j, \theta) \leq \max_{\theta' \in \Theta_j} \ell(T_j, \theta')$. Thus, this approach allows us to provide guarantees about the level of forgetting.

However, applying this idea directly to neural networks is intractable. In fact, as Knoblauch et al. (2020) noted, even finding the intersection between regions Θ_j and Θ_{j+1} of an arbitrary shape is an NP-hard problem. In order to bring the problem back to tractable regime, we try to instead find $\bar{\Theta} \subseteq \Theta$, where $\bar{\Theta}$ has a restricted shape that can be easily intersected. That is, we want to find:

$$\bar{\Theta}_j = \arg\max_{\bar{\Theta}} \|\bar{\Theta}\| \text{ satisfying } \bar{\Theta} \subseteq \Theta, \bar{\Theta} \in \mathcal{G}, \tag{3}$$

where $\mathcal{G}$ is a family of sets with required properties. For example, in order to solve the problem of NP-hard intersections we could use $\mathcal{G}$ as the set of convex polytopes. However, this restriction is not enough to make optimization of Eq. (2) tractable. Instead, we restrict this family even further, and we set $\mathcal{G}$ to be a set of hyperrectangles, which, as we will show, allows us to develop a viable algorithm for training neural networks in the continual learning setting. With this formulation, we lose some plasticity as we are not able to represent points in the region $\Theta_j \setminus \bar{\Theta}_j$, but in the experimental section we show that the final model still has enough expressiveness to solve difficult tasks.

Using this formulation, we will now show how to apply the Hyperrectangle Training to continual learning problem with neural networks. In particular, we will demonstrate that for standard modern neural network architectures we can find a tractable and easily differentiable upper bound of $\max_{\theta \in \Theta} \ell(T_j, \theta)$ which will in turn allow us to perform optimization of Eq. (2).

Interval arithmetic for neural networks Let us consider the family of feed-forward neural networks trained for classification tasks. We assume that the neural network $\Phi(x; \theta)$ is defined as a sequence of transformations h_l for each of its L layers. The output $h_L(h_{L-1}(\dots h_1(x))) = z_L \in \mathbb{R}^N$ has N logits corresponding to N classes.

In Hyperrectangle Training we assume that weights of each transformation h_k are located arbitrarily in the hyperrectangle $[\underline{W}, \bar{W}]$. For a particular task T_j, the Cartesian product of all intervals $[\underline{W}_1, \bar{W}_1] \times [\underline{W}_2, \bar{W}_2] \times \dots [\underline{W}_L, \bar{W}_L]$ forms the hyperrectangle Θ_j. With this setting we aim to find an upper bound on $\max_{\theta \in \Theta} \ell(T_j, \theta)$.

Observe that for a given input x and particular interval layer $[\underline{W}, \bar{W}]$ we can bound the possible outputs of that layer. As such, we can bound not only the weights of our model but also the activations. Thus, we will use interval arithmetic as our main tool to formalize this problem.

Interval arithmetic (Dahlquist & Björck, 2008)(Chapter 2.5.3) is based on the operations on segments. Let us assume A and B are numbers expressed as intervals. For all $\bar{a}, \underline{a}, \bar{b}, \underline{b} \in \mathbb{R}$ where $A = [\underline{a}, \bar{a}]$, $B = [\underline{b}, \bar{b}]$, we can define operations such as (Lee, 2004):

- addition: $[\underline{a}, \bar{a}] + [\underline{b}, \bar{b}] = [\underline{a} + \underline{b}, \bar{a} + \bar{b}]$
- multiplication: $[\underline{a}, \bar{a}] * [\underline{b}, \bar{b}] = [\min(\underline{a} * \underline{b}, \underline{a} * \bar{b}, \bar{a} * \underline{b}, \bar{a} * \bar{b}), \max(\underline{a} * \underline{b}, \underline{a} * \bar{b}, \bar{a} * \underline{b}, \bar{a} * \bar{b})]$

Thus, we can use interval arithmetic to perform affine transformations, which in turn allows us to implement fully-connected and convolutional layers of neural networks. Let us consider the interval version of the classical dense layer:

$$h(x) = Wx + b.$$

In Hyperrectangle Training a vector of weights is a vector of intervals and consequently the output of the dense layer is also a vector of intervals:

$$[\underline{h}(x), \bar{h}(x)] = [\underline{W}, \bar{W}][x, x].$$

InterContiNet is defined by a sequence of transformations h_l for each of its L layers. That is, for an input $z_0 = [x, x]$, we have

$$[\underline{z}_l, \bar{z}_l] = h_l([\underline{z}_{l-1}, \bar{z}_{l-1}]) \text{ for } l = 1, \dots, L.$$

The output $[\underline{z}_L, \bar{z}_L]$ has N interval logits corresponding to N classes.

Propagating bounds through any element-wise monotonic activation function (e.g., ReLU, tanh, sigmoid) is trivial. Concretely, if h_l is an element-wise non-decreasing function, we have:

$$\underline{z}_l = h_l(\underline{z}_{l-1}), \qquad \bar{z}_l = h_l(\bar{z}_{l-1}).$$

As such, we can use interval arithmetic to push any given input x through the whole network and obtain outputs of N logits intervals corresponding to N classes $[\underline{z}_L, \bar{z}_L]$. In other words, we can find $z_L(x; \Theta_j) = [\underline{z}_L, \bar{z}_L]$ such that $\phi(x; \theta) \in z_L(x; \Theta_j)$ for any $\theta \in \Theta_j$.

Upper bound of maximum loss over a region Thanks to the interval arithmetic, we can now find an upper-bound of $\max_{\theta \in \Theta} \ell(T_j, \theta)$ which is efficient to compute.

Theorem 3.1. *Let Φ be a neural network, Θ be a region of parameter space of that neural network, and $z_L(x, \Theta)$ be a function returning a hyperrectangle such that $\Phi(x, \theta) \in z_L(x, \Theta)$ for any $\theta \in \Theta$. Define worst-case interval-based loss as: $\hat{l}(x, y; \Theta) = l(\hat{z}_L, y)$, where $\hat{z}_L$ is a vector with each element defined as:*

$$\hat{z}_L^{(i)} = \begin{cases} \bar{z}_L^{(i)}, & \textit{for } y \neq y, \\ \underline{z}_L^{(y)}, & \textit{otherwise} . \end{cases}$$

Then $\hat{l}(x, y; \Theta) \geq \max_{\theta \in \Theta} l(x, \theta)$.

Proof. First, we will show that that $\hat{z}_L$ is the maximum of the cross-entropy loss, i.e. $\hat{z}_l = \arg\max_{z \in z_L(x,\Theta)} l(z, y)$. From the definition of $\hat{l}$, for any $\tilde{z} \in z_L(x, \Theta)$, for the worst-case prediction of the true class (when $i = y$) we have $\tilde{z}^{(y)} \geq \hat{z}_L^{(y)}$. Similarly, for the incorrect class ($i \neq y$) we have $\tilde{z}^{(i)} \leq \hat{z}_L^{i}$. When we consider coordinate connected with correct label $y = i$ cross-entropy will be larger than any other elements from interval since we return a lower prediction on the correct class. Analogically, when we consider coordinate connected with an incorrect label $i \neq y$ any element larger than the minimum of the interval, we obtain lower cross-entropy since we increase the probability of false classes. From that, it is evident that $\hat{z}_L$ is the maximum argument in this hyperrectangle.

Now, denote the set of possible logits produced by the neural network over the region Θ as $\Phi(x) = \{\phi(x, \theta) \mid \theta \in \Theta\}$. Since we know that $\phi(x, \theta) \in z_L(x, \Theta)$, we see that $\Phi(x) \subseteq z_L(x, \Theta)$. Since $\hat{z}$ is the maximum over the set $z_L(x, \Theta)$, then it must also be the maximum over its subset $\Phi(x)$. □

This gives us the worst-case scenario over a parameter region for a particular example. It is now straightforward to define $\hat{\ell}(T_j, \Theta) = \frac{1}{|T_j|} \sum_{(x,y) \in T_j} \hat{l}(x, y; \Theta)$ to be the worst-case scenario for the whole task. Now we can prove that this $\hat{\ell}$ is an upper bound of the whole continual learning objective.

Theorem 3.2. *Assume a sequential learning problem described by (2). Then during task j for any $k < j$, we have $\hat{\ell}(T_k, \Theta_j) \geq \max_{\theta' \in \Theta_j} \ell(T_k, \theta') \geq \max_{\theta' \in \Theta_k} \ell(T_k, \theta')$*

Proof. The first inequality follows from Theorem 3.1:

$$\begin{aligned} \hat{\ell}(T_j, \Theta) &= \frac{1}{|T_j|} \sum_{(x,y) \in T_j} \hat{l}(x, y; \Theta) \\ &\geq \frac{1}{|T_j|} \sum_{(x,y) \in T_j} \max_{\theta \in \Theta} l(x, y; \theta) \\ &\geq \max_{\theta \in \Theta} \frac{1}{|T_j|} \sum_{(x,y) \in T_j} l(x, y; \theta) \\ &= \max_{\theta \in \Theta} \ell(T_j, \theta) \end{aligned} \tag{4}$$

The second inequality follows directly from the fact that $\Theta_j \subseteq \Theta_k$. □

Analogously to $\hat{\ell}$, we can define $\hat{\text{acc}}(T_j, \Theta_j)$ to be the interval lower bound on minimum accuracy for task T_j over a region of parameter space Θ_j, i.e. $\hat{\text{acc}}(T_j, \Theta_j) \leq \min_{\theta \in \Theta_j} \text{acc}(T_j, \theta)$. Since for each $k > j$, $\Theta_k \subseteq \Theta_j$, we can guarantee that throughout the training the accuracy for j-th task will not fall below $\hat{\text{acc}}(T_j, \Theta_j)$.

To summarize, we have shown that by using Hyperrectangle Training we obtain guarantees on non-forgetting since the interval from task j is inside segments from the previous $j - 1$-th task. Moreover, we can solve such a problem in polynomial time, since we restrict our consideration to hyperrectangles. Finally, in Theorem 3.2 we demonstrate that the bounds on logits provided by interval arithmetic on neural networks can be used to calculate an efficient upper bound on the whole continual learning objective.

The result presented here is relevant to Knoblauch et al. (2020) who have shown that classical continual learning problem is in general NP-hard because of the highly irregular shapes of the viable regions of parameters. In Section B we present a more rigorous treatment of the above derivation, using the setting and assumptions proposed in Knoblauch et al. (2020).

4. InterContiNet

In the previous section, we have shown that by considering the intervals of activations in a neural network we can derive an upper bound of the maximum loss over a hyperrectangle. Now, we will present a model which minimizes this upper

Algorithm 1 InterContiNet training procedure for a given task

```
Input: model trained on the previous task (weights $W^*$, radii $\varepsilon^*$), current task $T_j$.
Reparameterize $W, \varepsilon$ using Eq. (5).
for epoch in $1 \dots$ center_epochs do
    Update $\mu$ by minimizing $\ell(T_j, \Theta)$      {Train centers}
end for
Initialize $\varepsilon$ as largest possible within the previous interval.
for epoch in $1 \dots$ radii_epochs do
    Update $\nu$ by minimizing $\hat{\ell}(T_j, \Theta)$      {Train radii}
    if $\hat{\text{acc}} \geq \text{acc} \cdot \text{acc\_thresh}$ then
        break
    end if
end for
return $W$, $\varepsilon$
```

bound in order to approximate the optimization presented in Eq. (2).

Parametrization We start by taking a standard feed-forward neural network Φ with parameters θ, but instead of considering points in the parameter space, we consider regions Θ. In particular, we describe the k-th parameter of the network with the center W_k and the interval radius ε_k. Thus, we consider the region:

$$[\underline{W}_k, \bar{W}_k] = [W_k - \varepsilon_k, W_k + \varepsilon_k],$$

With this formulation, we can still use this network as a standard non-interval model by only using the center weights W_k, which will, in turn, produce only the center activations for each layer. At the same time, we can also keep track of the interval $[\underline{z}_l, \bar{z}_l]$ by using the interval arithmetic. These operations are independent of each other, so as explained in Gowal et al. (2018) we can implement them efficiently for GPUs by calculating the center prediction and the intervals in parallel. In practice, we reimplement the basic blocks of neural networks (fully-connected layers, convolutional layers, activations, pooling, etc.) in order to handle interval inputs and interval weights.

With this formulation, we are able to implement most operations in contemporary standard neural network architectures. However, one important exception is batch normalization which uses running statistics of activations in inference mode. Since it is impossible to foresee how these statistics will change in the future, providing reliable forgetting guarantees with interval arithmetic is infeasible. As such, we restrict ourselves to simpler architectures without batch normalization.

During the training of the second and every subsequent task, we need to make sure that the intervals for the current task are contained within intervals from the previous task. Starting from the second task, we keep the previous interval centers W_k^* and radii ε_k^*. Then, we reparameterize the training as:

$$\begin{aligned} W_k &= W_k^* + \tanh(\mu_k) \cdot \varepsilon_k^* \\ \varepsilon_k &= \sigma(\nu_k) \cdot \min((W_k^* + \varepsilon_k^*) - W_k, \\ &\qquad W_k - (W_k^* - \varepsilon_k^*)). \end{aligned} \tag{5}$$

Observe that we cannot simply set the radius to be $\sigma(\nu_k) \cdot \varepsilon_k^*$ as depending on the position on the center W_k the interval $[W_k - \varepsilon_k, W_k + \varepsilon_k]$ might not be fully contained inside the previous task interval $[W_k^* - \varepsilon_k^*, W_k^* + \varepsilon_k^*]$. We train μ_k and ν_k instead of directly optimizing W_k and ε_k.

Training When training InterContiNet in a continual learning setting, we divide training of each task T_j into two phases. In the first phase, we focus on finding the interval centers W_k and keep the radii ε_k frozen. We do this by simply minimizing the cross-entropy loss, as done in standard classification neural network training. In the second phase, we freeze the centers W_k and initialize ε_k with the maximal value that still fits within the previous interval[1]. Then, we minimize the upper-bound loss $\hat{\ell}(T_j, \Theta_j)$ by optimizing ε_k. Optimizing this upper bound indefinitely would lead to minimizing ε_k to zeroes and producing degenerated intervals. To prevent this, we introduce an interpretable hyperparameter acc_thresh which allows us to choose the fraction of examples in the current task that is guaranteed to be classified correctly for the rest of the training. For that purpose, we use the worst-case accuracy $\hat{\text{acc}}$ described in Section 3 and minimize the worst-case loss $\hat{l}$ until the condition $\hat{\text{acc}} \geq \text{acc} \cdot \text{acc\_thresh}$ is satisfied, with acc being the train accuracy on the current task. We use running averages on the last few batches to efficiently estimate $\hat{\text{acc}}$ and acc. As such, acc_thresh is the variable that controls the stability-plasticity trade-off in our model. A high value of acc_thresh will guarantee less forgetting in the end, but will also result in a smaller parameter region Θ_j, which will be used to constrain the parameter search in the next task. Algorithm 1 shows the whole training procedure for a single task.

Analysis We believe that realistic computational and memory constraints are an important part of the continual learning setting. In terms of the memory requirements, InterContiNet needs to remember the centers W_K^* and radii ε_k^* from the previous task. Thus, assuming a network with D parameters, we need to keep additional $2 \cdot D$ values in memory. This is the same memory constraint as for most of our baselines, e.g. online EWC, which remembers the weights and their importance for the last task.

[1]For the first task the maximum radius is a hyperparameter to be set.

In terms of computational constraints, propagating both activation centers and intervals through the network requires approximately 3 times as many FLOPs as propagating just a single point. However, the main part of the training is the center optimization phase, that has no overhead as compared to the base network since we only need to push forward a single point (the centers). The interval propagation is only needed in the radii optimization phase that is a small part of the overall training. Thus, in practice it is on a similar order of complexity as, e.g. computing the Fisher information matrix in EWC. Finally, in practice even in the interval propagation phase the training time does not change significantly, as these operations are automatically parallelized on GPUs when using frameworks such as PyTorch.

5. Experiments

To verify the empirical usefulness of our method, we test it in three standard continual learning scenarios (Hsu et al., 2018; van de Ven & Tolias, 2019): incremental task, incremental domain, and incremental class. In the incremental task setting, we create a separate output head for each task and only train a single head per task along with the base network. During inference, the head appropriate for the given task is selected. The incremental domain and incremental class scenarios are more challenging, as they only use a single head and the task identity is not provided during inference. In our case, this single head is created with either 2 (incremental domain) or 10 output classes (incremental class).

We run the experiments on four datasets commonly used for continual learning: MNIST, FashionMNIST, CIFAR-10 and CIFAR-100 split into sequences of tasks. For MNIST and FashionMNIST, we use a standard MLP architecture with 2 hidden layers of size 400 as previously evaluated in (Hsu et al., 2018). For CIFAR-10 and CIFAR-100 we use the fairly simple AlexNet architecture (Krizhevsky et al., 2012). This choice is dictated by the infeasibility of implementing interval version of batch normalization which is present in most standard convolutional network architectures. In our experiments, we use the Avalanche (Lomonaco et al., 2021) library to facilitate easier and fairer comparisons. The code is available at `https://github.com/gmum/InterContiNet`.

We compare InterContiNet with a number of regularization-based approaches typically used in continual learning settings, i.e. EWC, Synaptic Intelligence, MAS, and LwF. As additional baselines, we present vanilla sequential training with SGD and Adam, and a simple L_2 regularization between new and old model parameters (i.e. all parameters have the same importance). Further training details are available in Appendix A.

Table 1. The average accuracy across all five tasks of the split MNIST protocol, evaluated after learning the whole sequence. Each value is the average of five runs (with standard deviations).

Method	Incremental task	Incremental domain	Incremental class
SGD	96.27 ± 0.38	64.58 ± 0.26	19.01 ± 0.04
Adam	95.53 ± 3.16	59.32 ± 1.08	19.74 ± 0.01
L2	96.31 ± 0.41	72.21 ± 0.17	18.88 ± 0.18
EWC	97.01 ± 0.13	76.90 ± 0.41	18.90 ± 0.06
oEWC	97.01 ± 0.13	77.02 ± 0.51	18.89 ± 0.07
SI	96.19 ± 0.63	80.62 ± 0.17	17.94 ± 0.57
MAS	96.52 ± 0.14	**84.41 ± 0.42**	17.38 ± 4.19
LwF	97.03 ± 0.05	82.76 ± 0.17	**49.37 ± 0.68**
InterContiNet	**98.93 ± 0.05**	77.77 ± 1.24	40.73 ± 3.26
Offline	99.74 ± 0.03	99.03 ± 0.04	98.49 ± 0.02

Table 2. The average accuracy across all five tasks of the split FashionMNIST protocol, evaluated after learning the whole sequence. Each value is the average of five runs (with standard deviations).

Method	Incremental task	Incremental domain	Incremental class
SGD	92.22 ± 3.06	82.77 ± 0.44	19.91 ± 0.01
Adam	88.13 ± 5.64	78.48 ± 0.47	19.96 ± 0.01
L2	97.36 ± 0.17	**92.65 ± 0.09**	26.91 ± 1.23
EWC	97.53 ± 0.15	92.12 ± 0.18	19.90 ± 0.01
oEWC	96.70 ± 0.57	88.83 ± 0.30	19.87 ± 0.02
SI	97.00 ± 0.25	91.45 ± 0.07	19.97 ± 0.34
MAS	97.43 ± 0.14	91.74 ± 0.19	10.00 ± 0.00
LwF	98.10 ± 0.07	88.63 ± 0.12	**39.51 ± 1.45**
InterContiNet	**98.37 ± 0.06**	**92.65 ± 0.40**	35.11 ± 0.02
Offline	97.98 ± 0.05	96.39 ± 0.06	82.54 ± 0.13

5.1. MNIST and CIFAR Benchmarks

Results presented in Tables 1 and 2 indicate that in the incremental task setup InterContiNet has better average accuracy across all five tasks than all the other continual learning baselines. We hypothesize that a multi-head setup facilitates our 2-phase training procedure, as the output heads in the last layer are independent and do not have to be constrained inside intervals. In the incremental domain setup, our method has a similar performance to EWC. However, the advantage of InterContiNet is that we can guarantee a maximal level of forgetting on the past tasks when training on new data.

The most significant difference is visible in the incremental class scenario, where LwF and InterContiNet perform much better than typical regularization-based approaches. We can partly explain this behavior with the particular setup, where the final number of output classes is known at the onset of the experiment. While such information induces an indirect regularization effect on the softmax layer in LwF and InterContiNet, it has smaller effect on other techniques. This

Table 3. The average accuracy across all five tasks of Split CIFAR-10, evaluated after training AlexNet on the whole sequence. Each value is the average of five runs (with standard deviations).

Method	Incremental task	Incremental domain	Incremental class
SGD	64.74 ± 8.53	69.22 ± 0.55	15.56 ± 5.07
EWC	67.33 ± 7.11	70.26 ± 0.66	11.83 ± 4.10
oEWC	64.59 ± 7.97	65.97 ± 8.97	15.49 ± 5.01
SI	67.26 ± 7.77	69.69 ± 0.73	17.38 ± 4.13
MAS	66.20 ± 9.29	72.54 ± 0.59	11.79 ± 4.01
LwF	**93.03** ± 0.28	**76.97** ± 0.91	13.89 ± 5.33
InterContiNet	72.64 ± 1.18	69.48 ± 1.36	**19.07** ± 0.15
Offline	93.11 ± 0.18	90.54 ± 0.32	82.72 ± 0.09

dichotomy highlights the fundamental difference between LwF's knowledge distillation approach and all the other baselines, which are prior-focused.

Finally, we extend our investigation to CIFAR-10 with the AlexNet. We choose this architecture as an example of a convolutional network without batch normalization, which is problematic in our setting, as we explain later in Section 6. As shown in Table 3, in this setting InterContiNet still performs comparably to other parameter regularization methods. At the same time, we observe that LwF tends to significantly outperform other methods, especially in the incremental task setting. We hypothesize that the regularization through functional penalties (KL divergence between outputs) rather than direct parameter regularization works much better for this particular problem (Oren & Wolf, 2021). In the end, we show that InterContiNet is able to perform similarly to other methods in the same family while providing guarantees.

5.2. Analysis of the Parameter Region Size

Although the stability-plasticity dilemma is a crucial problem in all CL methods, in InterContiNet its impact can be directly observed by investigating the region of valid parameters Θ. The bigger the region, the higher the plasticity, as we are able to use more parameter combinations and thus represent more functions. At the same time, in order to achieve stability and get guarantees on forgetting, we need to restrict the size of this region. Here, we investigate the question of whether the interval training will at some point lead to an area of zero volume, resulting in a state without any plasticity and effectively stopping the learning[2].

In practice, we control the pace at which the volume decreases with hyperparameters, e.g. by choosing the worst-case accuracy threshold – if we set this threshold high, the

[2]We note that this risk also occurs for other CL methods (e.g. accumulating high EWC penalties which prohibit learning or reducing the percentage of trainable parameters in PackNet to zero).

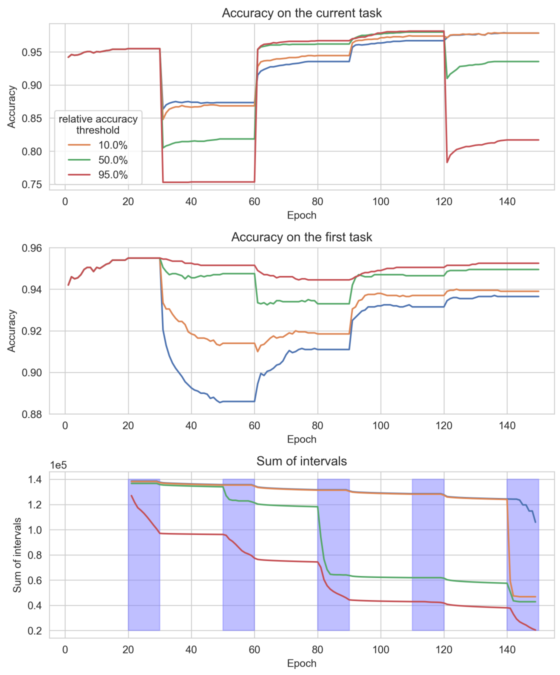


Figure 3. InterContiNet trained on split-Fashion MNIST in the incremental domain setting, with different values of the $\mathrm{acc\_thresh}$ hyperparameter. We show accuracy on the current task (top, representing plasticity) and the first task (middle, representing stability) to show how the hyperparameter choices affect the final results. Additionally, the bottom plot shows how the valid parameter region changes throughout the training – highlighted areas denote the radii optimization phase, which shrinks this region most effectively.

region collapses quicker. Here, we conduct additional studies to better understand how the threshold hyperparameter impacts the results and the dynamics of the region of valid parameters. In Figure 3 we present the accuracy curves for training InterContiNet on FashionMNIST in the incremental domain setting, using different values of the accuracy threshold hyperparameter. This threshold controls the percentage of the correctly classified examples that we guarantee will not be forgotten throughout the training. For example, in Figure 3 the method obtains 95.3% accuracy at the end of the first task. If we set $\mathrm{acc\_thresh} = 95\%$, then we can guarantee that at no point during training the accuracy on the first task will be lower than $95\% \cdot 95.3\% \approx 90.5\%$.

We see that this hyperparameter indeed controls the plasticity-stability trade-off, with larger values contributing to less forgetting, but also less plasticity on the subsequent tasks. Note that in practice the obtained accuracy is much higher than the one guaranteed by the worst-case accuracy. For example, even with $\mathrm{acc\_thresh} = 0.05$ we are able to maintain first-task accuracy above 90% through the whole training. However, this performance is not guaranteed and might deteriorate with further training.

In the bottom part of Figure 3, we plot how the valid pa-

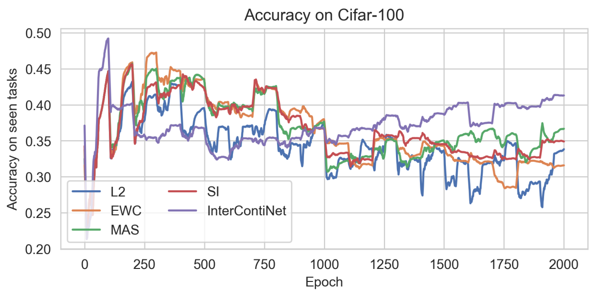


Figure 4. Results for parameter regularization methods for split-CIFAR-100 with 20 tasks, averaged over 3 seeds. InterContiNet is still able to learn new tasks at the end of the training.

rameter region changes throughout the training. Since the volume can get easily degenerated (e.g. if we prohibit even one parameter from changing), we summarize it by reporting the sum of all radii. Confirming the intuition, the valid region for models with higher $\mathrm{acc\_thresh}$ shrinks much faster, while for lower thresholds the impact is not as noticeable. In the end, we observe that even in the most restrictive setting, there is still room for learning.

Finally, we extend the experiments on the collapse of the valid parameter region to a setting with a higher number of tasks. We use split-CIFAR-100 with 20 tasks of 5 classes and focus on the incremental task scenario. The results in Figure 4 show that InterContiNet performs on similar scale as other parameter regularization methods and is still able to learn new tasks at the end of the training. We consider this as additional evidence that interval constraints do not necessarily lead to faster plasticity collapse than soft regularization. Additionally, we ran LwF in this setting and found out that it significantly outperforms the parameter regularization methods by a very wide margin. We show the additional results in Figure 5 in Appendix A. This finding is consistent with the split-CIFAR-10 results and recent analysis (Oren & Wolf, 2021), and is a strong motivation to consider functional regularization in future work.

6. Limitations & Future Work

Although reformulating continual learning as a sequential contraction of the model's parameter space creates a viable theoretical framework, InterContiNet has several limitations that would require further research to be addressed.

To begin with, interval arithmetic cannot be easily combined with batch normalization layers. As we cannot foresee the running statistics of activations at future inference, we cannot guarantee the bounds of the model's output. This limitation is problematic due to the ubiquitousness of batch normalization in contemporary neural network models. We plan to evaluate InterContiNet with different normalization schemes (e.g. layer (Ba et al., 2016) or group normalization (Wu & He, 2020)), where providing guarantees is still possible. Another option worth exploring is freezing the batch normalization statistics after the first task.

On the other hand, the no-forgetting guarantees are also limited in several ways. Firstly, we concentrate on the worst-case scenario, i.e. we maintain that the worst solution in the allowed parameter space will not get even worse. In practice, we evaluate the model at the interval centers, which can leave a lot of leeway for the accuracy to fluctuate above the defined threshold. Secondly, these guarantees are computed on the training set. Depending on the amount of training–test distribution shift, a non-negligible change in the model's response can be unaccounted for in our worst-case scenario.

Another drawback of hard constraints is the risk of reducing the viable parameter space to a zero-volume region when exposed to very long sequences of tasks or very deep architectures. While it is a very reasonable concern, the goal of our empirical evaluation was to show that InterContiNet's parameter space does not degenerate so rapidly in typical experimental setups. Nevertheless, based on Theorem 3.1, it is theoretically possible to devise a malicious dataset that would result in arbitrarily large forgetting or complete loss of plasticity. Lastly, our evaluation is limited to classification problems only. However, extensions to other types of problems (e.g. regression) could be made with simple adjustments to the loss function.

To summarize, in this work we proposed Hyperrectangle Training, a general approach to continual learning. We constrain the parameter search within the hyperrectangle representing the region of valid solutions for the previous task. We show that by applying interval arithmetic to operations present in standard neural networks in this setting we can derive a tractable and differentiable upper bound of the CL objective. Then we proposed and empirically evaluated InterContiNet, a CL algorithm that minimizes this upper bound to set hard limits on forgetting.

7. Acknowledgements

The work of M. Wołczyk, K. Piczak, J. Tabor, and T. Trzciński was supported by Foundation for Polish Science (grant no POIR.04.04.00-00-14DE/18-00) carried out within the Team-Net program co-financed by the European Union under the European Regional Development Fund. The work of P. Spurek was supported by the National Centre of Science (Poland) Grant No. 2019/33/B/ST6/00894. The work of T. Trzciński was supported by the National Centre of Science (Poland) Grant No. 2020/39/B/ST6/01511. The work of B. Wójcik was supported by the Priority Research Area DigiWorld under the program Excellence Initiative - Research University at the Jagiellonian University in Kraków.

References

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Ba, L. J., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Chakraverty, S. and Sahoo, D. M. Interval response data based system identification of multi storey shear buildings using interval neural network modelling. *Computer Assisted Methods in Engineering and Science*, 21(2):123–140, 2014.

Chakraverty, S. and Sahoo, D. M. Novel transformation-based response prediction of shear building using interval neural network. *Journal of Earth System Science*, 126(3): 32, 2017.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with A-GEM. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Dahlquist, G. and Björck, Å. *Numerical methods in scientific computing, volume I.* SIAM, 2008.

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

French, R. M. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Goodfellow, I. J., Mirza, M., Da, X., Courville, A. C., and Bengio, Y. An empirical investigation of catastrophic forgeting in gradient-based neural networks. In Bengio, Y. and LeCun, Y. (eds.), *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T., and Kohli, P. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

Hsu, Y., Liu, Y., and Kira, Z. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*, 2018.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

Knoblauch, J., Husain, H., and Diethe, T. Optimal continual learning has perfect memory and is NP-hard. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

Lee, K. H. *First course on fuzzy theory and applications*, volume 27. Springer Science & Business Media, 2004.

Lee, S., Kim, J., Jun, J., Ha, J., and Zhang, B. Overcoming catastrophic forgetting by incremental moment matching. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017.

Liu, X., Masana, M., Herranz, L., van de Weijer, J., López, A. M., and Bagdanov, A. D. Rotate your networks: Better weight consolidation and less catastrophic forgetting. *Proceedings of the 24th International Conference on Pattern Recognition (ICPR)*, 2018.

Lomonaco, V., Pellegrini, L., Cossu, A., Carta, A., Graffieti, G., Hayes, T. L., Lange, M. D., Masana, M., Pomponi, J., van de Ven, G., Mundt, M., She, Q., Cooper, K., Forest, J., Belouadah, E., Calderara, S., Parisi, G. I., Cuzzolin, F., Tolias, A., Scardapane, S., Antiga, L., Amhad, S., Popescu, A., Kanan, C., van de Weijer, J., Tuytelaars, T., Bacciu, D., and Maltoni, D. Avalanche: an end-to-end library for continual learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2nd Continual Learning in Computer Vision Workshop, 2021.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Mallya, A., Davis, D., and Lazebnik, S. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Masse, N. Y., Grant, G. D., and Freedman, D. J. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 115:E10467 – E10475, 2018.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Mirzadeh, S., Farajtabar, M., Görür, D., Pascanu, R., and Ghasemzadeh, H. Linear mode connectivity in multitask and continual learning. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021.

Morawiecki, P., Spurek, P., Śmieja, M., and Tabor, J. Fast and stable interval bounds propagation for training verifiably robust models. *arXiv preprint arXiv:1906.00628*, 2019.

Oren, G. and Wolf, L. In defense of the learning without forgetting for task incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

Proszewska, M., Mazur, M., Trzciński, T., and Spurek, P. HyperCube: Implicit field representations of voxelized 3D models. *arXiv preprint arXiv:2110.05770*, 2021.

Ratcliff, R. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. Experience replay for continual learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Sahoo, D. M. and Chakraverty, S. Structural parameter identification using interval functional link neural network. In *Recent Trends in Wave Mechanics and Vibrations*, pp. 139–150. Springer, 2020.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

van de Ven, G. M. and Tolias, A. S. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

van de Ven, G. M., Siegelmann, H. T., and Tolias, A. S. Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*, 11, 2020.

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., and Farhadi, A. Supermasks in superposition. *arXiv preprint arXiv:2006.14769*, 2020.

Wu, Y. and He, K. Group normalization. *International Journal of Computer Vision (IJCV)*, 128(3):742–755, 2020.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

A. Training Details

Data We use MNIST, FashionMNIST, CIFAR-10, and CIFAR-100 datasets with original training and testing splits. We do not use any form of data augmentation or normalization. We split the datasets into 5 tasks containing data from classes: [0, 1], [2, 3], [4, 5], [6, 7], [8, 9]. We split CIFAR-100 into 20 tasks with 5 classes.

Architecture The baseline model for MNIST and FashionMNIST is a MLP with 2 hidden layers of 400 units and a linear output layer (single- or multi-head). We use ReLU activations between layers. For CIFAR-10 and CIFAR-100 we use the AlexNet (Krizhevsky et al., 2012) architecture adapted to 32×32 inputs. For InterContiNet we simply replace standard fully-connected layers with its interval variants, with the linear layers of the heads in the incremental task setting being an exception. That is, those layers also accept the lower and upper bounds as input, but we do not add μ and ν for them, as the weights for these layers are task-specific.

Hyperparameters In the MNIST and Fashion-MNIST experiments we use batch size of 128, 30 epochs of training for each task (i.e. 150 in total) and vanilla cross entropy loss. In the offline (upper bound approximation) learning experiment we use a single combined task with 100 epochs of training. We repeat each experiment 5 times with different seeds to get the mean and the standard deviation estimates.

In the SGD baseline and every continual learning baseline we use the SGD optimizer without momentum. The learning rate is set to 0.001 for MNIST and FashionMNIST experiments, and 0.01 for split-CIFAR-10 and split-CIFAR-100. In CIFAR experiments we additionally decrease the learning rate 10x in the 50th and the 90th epoch.

Tables 4, 5, 6 contain chosen hyperparameters for MNIST, CIFAR-10 and CIFAR-100 respectively, for the baseline methods.

Table 4. Hyperparameters for the baseline methods used in the MNIST and FashionMNIST experiments.

Method	Hyperparameter	Value
L2	λ	0.1
EWC	λ	2048
Synaptic Intelligence	λ	2048
MAS	λ	1
LwF	temperature	0.5
	α	0.5

Table 5. Hyperparameters for the baseline methods used in the CIFAR-10 experiments.

Method	Hyperparameter	Value
L2	λ	0.05
EWC	λ	0.05
Synaptic Intelligence	λ	1.
MAS	λ	0.005
LwF	temperature	1.
	α	1.

Table 6. Hyperparameters for the baseline methods used in the CIFAR-100 experiments.

Method	Hyperparameter	Value
L2	λ	0.005
EWC	λ	0.0005
Synaptic Intelligence	λ	0.1
MAS	λ	0.0005
LwF	temperature	1.
	α	0.5

Since $\hat{\ell}$ and ℓ have values differing by orders of magnitude, we use a separate learning rates for training centers W and

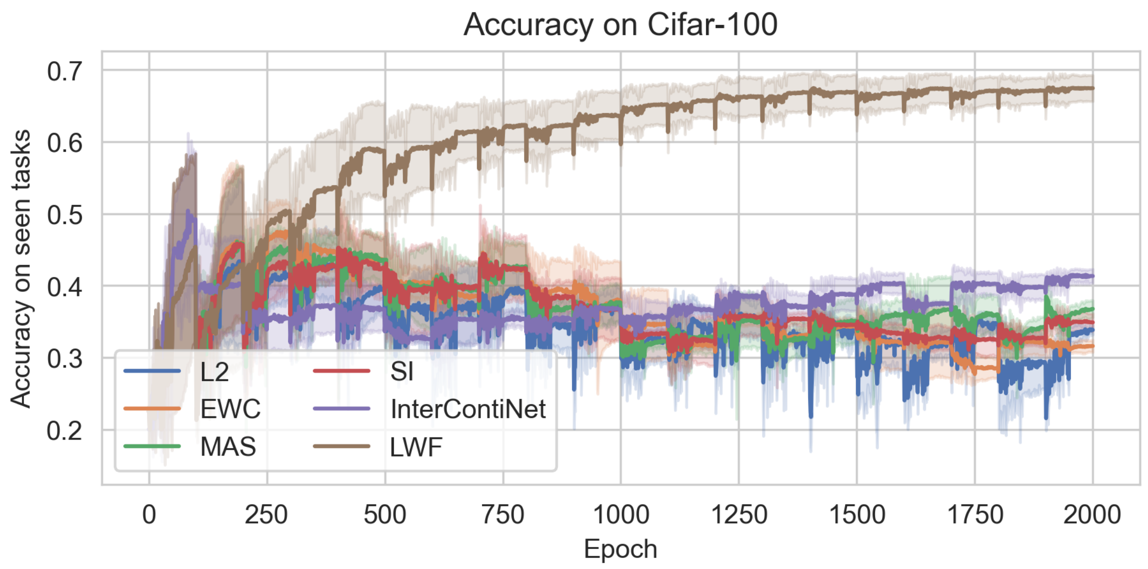


Figure 5. Full results for CIFAR-100, computed from 3 seeds. The confidence intervals represent standard deviation.

radii ε. We tune these two learning rates and the $\mathrm{acc\_thresh}$ hyperparameter and list them in Table 7. Similarly as in the baselines, we use SGD without momentum. After each task the ν parameters are reset to the value of 5.0.

Table 7. Hyperparameters for InterContiNet on each dataset and setting combination.

Dataset	Setting	acc_thresh	Center optimization LR	Radii Optimization LR	Initial radius
MNIST	IT	0.9	1.0	100	1
	ID	0.8	1.0	1000	1
	IC	0.8	0.001	1.0	1
FashionMNIST	IT	0.9	0.001	100	1
	ID	0.8	0.001	10	1
	IC	0.9	0.001	0.1	1
CIFAR-10	IT	0.1	0.1	0.1	0.1
	ID	0.1	0.1	10^{-10}	0.1
	IC	0.1	0.1	10^{-20}	0.1
CIFAR-100	IT	0.05	0.001	100	0.5

Additional results In Figure 5 we show full results for split CIFAR-100, including LwF and standard deviation confidence regions.

B. Formal Presentations of the Results

In (Knoblauch et al., 2020) authors develop a theoretical approach that explains why catastrophic forgetting is a challenging problem. Furthermore, the authors show that optimal Continual Learning algorithms generally have to solve a NP-HARD problem. Thanks to interval arithmetic we consider only hyperrectangles. By constraining the CL problem to interval architecture we reduce the complexity to a polynomial one. In the experimental section, we show that such a solution also gives excellent practical results.

Roughly speaking, the main problem with the continual learning setting is that areas in the parameter space that guarantee each task's performance can have arbitrary shape, see Fig. 1. Thus, our primary goal is to detect weight in the intersection of such regions. Even simple linear models together with an intuitively appealing upper bound on the prediction error as optimality criterion are NP-HARD (see (Knoblauch et al., 2020)[Example 1]).

In our framework, we constrain the areas in the parameter space which guarantee the solution of each task to hyperrectangles, see Fig. 1(b) (bottom). Thus, the intersection of hyperrectangles is a hyperrectangle, and we can produce that intersection in a polynomial time.

For convenience of the reader we introduce the notation from (Knoblauch et al., 2020). We deal with random variables $\mathbf{X_t}$, $\mathbf{Y_t}$. Realizations of the random variable $\mathbf{X_t}$ live on the input space $\mathcal{X}$ and provide information about the random outputs $\mathbf{Y_t}$ with realizations on the output space $\mathcal{Y}$. Throughout, $P(A)$ denotes the collection of all probability measures on A.

Definition B.1. (Tasks). For a number $T \in \mathbb{N}$ and random variables $(X_t, Y_t)_{t=1}^T$ defined on the same spaces X and Y, the random variable (X_t, Y_t) is the t-th task, and its probability space is $(X_t \times Y_t, \Sigma, \mathbb{P}_t)$, where Σ is a σ-algebra and $\mathbb{P}_t$ a probability measure on $X_t \times Y_t \subset \mathcal{X} \times \mathcal{Y}$.

Given a sequence of samples from task-specific random variables, a CL algorithm sequentially learns a predictor for Y_t given X_t. This means that there will be some hypothesis class F_Θ consisting of conditional distributions which allow (probabilistic) predictions about likely values of Y_t.

Definition B.2. (CL hypothesis class). The CL hypothesis class F_Θ is parameterized by Θ: For any $f \in F_\Theta$, there exists a $\theta \in \Theta$ so that $f_\theta = f$. More precisely, $F_\Theta \subset P(\mathcal{Y})^{\mathcal{X}}$ if the task label is not conditioned on. Alternatively, $F_\Theta \subset P(\mathcal{Y})^{X \times \{1,2,\ldots T\}}$ if the label is conditioned on.

Definition B.3. (Continual Learning). For a CL hypothesis class $F_\Theta, T \in \mathbb{N}$ and any sequence of probability measures $\{\hat{\mathbb{P}}_t\}_{t=1}^T$ such that $\hat{\mathbb{P}}_t \in \mathcal{P}(\mathcal{Y}_t)^{\mathcal{X}_t} \subseteq P(Y)^{\mathcal{X}}$, CL algorithms are specified by functions

$$\hat{\mathcal{A}}_I : \Theta \times I \times \mathcal{P}(\mathcal{Y})^{\mathcal{X}} \to I$$
$$\hat{\mathcal{A}}_\Theta : \Theta \times I \times P(\mathcal{Y})^{\mathcal{X}} \to \Theta,$$

where I is some space that may vary between different CL algorithms. Given A_I and A_θ and some initializations θ_0 and I_0, CL defines a procedure given by

$$\begin{aligned} \theta_1 &= \hat{\mathcal{A}}_\Theta(\theta_0, I_0, \hat{\mathbb{P}_1}) \\ I_1 &= \hat{\mathcal{A}}_I(\theta_1, I_0, \hat{\mathbb{P}}_1) \\ \theta_2 &= \hat{\mathcal{A}}_\Theta(\theta_1, I_1, \hat{\mathbb{P}}_2) \\ I_2 &= \hat{\mathcal{A}}_I(\theta_2, I_1, \hat{\mathbb{P}}_2) \\ &\cdots \\ \theta_T &= \hat{\mathcal{A}}_\Theta(\theta_{T-1}, I_{T-1}, \hat{\mathbb{P}}_T) \\ I_T &= \hat{\mathcal{A}}_I(\theta_T, I_{T-1}, \hat{\mathbb{P}}_T) \end{aligned}$$

In (Knoblauch et al., 2020) authors introduce very flexible formalism. In particular, all that it needs is an arbitrary binary-valued optimality criterion $\mathcal{C}$, whose function is to assess whether or not information of a task has been retained ($\mathcal{C} = 1$) or forgotten ($\mathcal{C} = 0$). According to this formalism, a CL algorithm avoids catastrophic forgetting (as judged by the criterion $\mathcal{C}$) if and only if its output at task t is guaranteed to satisfy $\mathcal{C}$ on all previously seen tasks. In this context, different ideas about the meaning of catastrophic forgetting would result in different choices for $\mathcal{C}$. As we will analyze CL with the tools of set theory, it is also convenient to define the function SAT, which maps from task distributions into the subsets consisting of all values in Θ which satisfy the criterion $\mathcal{C}$ on the given task.

Definition B.4. For an optimality criterion $\mathcal{C} : \Theta \times \mathcal{P}(\mathcal{X} \times \mathcal{Y}) \to \{0, 1\}$ and a set $\mathcal{Q} \subseteq \mathcal{P}(\mathcal{X} \times \mathcal{Y})$ of task distributions, the function $SAT : \mathcal{P}(\mathcal{X} \times \mathcal{Y}) \to 2^\Theta$ defines the subset of Θ which satisfies $\mathcal{C}$ and is given by

$$SAT(\hat{\mathbb{P}}) = \{\theta \in \Theta : \mathcal{C}(\theta, \hat{\mathbb{P}}) = 1\}.$$

The collection of all possible sets generated by SAT is

$$SAT_\mathcal{Q} = \{SAT(\hat{\mathbb{P}}) : \hat{\mathbb{P}} \in \mathcal{Q}\}$$

and the collection of finite intersections from SATQ is

$$SAT_\cup = \{\cup_{i=1}^t A_i : A_i \in SAT_\mathcal{Q}, 1 \le i \le t \text{ and } 1 \le t \le T, T \in \mathbb{N}\}.$$

Now we are ready to introduce our theorem. We assume that for a task, we can train our model in intervals. In the experimental section, we show that such architecture can be efficiently trained.

Theorem B.5. *Take F_Θ to be the collection of MLP neural networks ϕ_θ*

with inputs on $\mathcal{X}$ and outputs on $\mathcal{Y} \subset \mathbb{R}$ linked through the coefficient vector $\theta \in \Theta$. Further, let $\mathcal{Q}$ be the collection of empirical measures

$$\hat{m}^t(y,x) = \frac{1}{n_t} \sum_{i=1}^{n_t} \delta_{(y_i^t, x_i^t)}(y,x)$$

whose $n_t \in \mathbb{N}$ atoms $\{(y_i^t, x_i^t)\}_{i=1}^{n_t}$ represent t-th task.

We assume that for all $i = 1, \ldots, n_t$ and $\varepsilon \geq 0$ exist $[\underline{W}_i, \bar{W}_i]$ such that for all $\theta \in [\underline{W}_i, \bar{W}_i]$ we have $|y_i^t - \phi_\theta(x_i^t)| \leq \varepsilon$.

Further, define for $\varepsilon \geq 0$ and all $\hat{\mathbb{P}} \in \mathcal{Q}$ the criterion

$$\mathcal{C}(\theta, \hat{\mathbb{P}}) = \begin{cases} 1, & \text{if } |y_i^t - \phi_\theta(x_i^t)| \leq \varepsilon \text{ and } \theta \in [\underline{W}_i, \bar{W}_i] \text{ for all } i = 1 \ldots n_t \\ 0, & \text{otherwise .} \end{cases}$$

Then,

$$\bigcap_{i=1}^{n_t} [\underline{W}_i, \bar{W}_i] \subset SAT(\hat{\mathbb{P}}),$$

Proof. Then, it is straightforward to see that

$$\begin{aligned} SAT(\hat{\mathbb{P}}) = \{\theta \in \Theta : y_i^t - \phi_\theta(x_i^t) \leq \varepsilon \text{ and } y_i^t - \phi_\theta(x_i^t) \geq -\varepsilon \text{ and } \theta \in [\underline{W}_i, \bar{W}_i], \text{ for all } i = 1 \ldots n_t\} = \\ = (\cap_{i=1}^{n_t} \{\theta \in \Theta : y_i^t - \phi_\theta(x_i^t) \leq \varepsilon\}) \cap (\cap_{i=1}^{n_t} \{\theta \in \Theta : y_i^t - \phi_\theta(x_i^t) \geq -\varepsilon\}) \cap (\cap_{i=1}^{n_t} [\underline{W}_i, \bar{W}_i]) \\ \supseteq \cap_{i=1}^{n_t} [\underline{W}_i, \bar{W}_i]. \end{aligned}$$

The intersection of hyperrectangles is a hyperrectangle. □

Because the rectangle intersection algorithm can be performed in polynomial time, our solution can be found in polynomial time as well. Furthermore, intersection of hyperrectangles from all tasks $\cap_{i=1}^{n_t} [\underline{W}_i, \bar{W}_i]$ is subset $SAT(\hat{\mathbb{P}})$, we have guaranteed not to forget previous tasks.