Optimizing Reusable Knowledge for Continual Learning via Metalearning

Julio Hurtado
Department of Computer Science
Pontificia Universidad Católica de Chile
jahurtado@uc.cl

Alain Raymond-Saez
Department of Computer Science
Pontificia Universidad Católica de Chile
afraymon@uc.cl

Alvaro Soto
Department of Computer Science
Pontificia Universidad Católica de Chile
asoto@ing.puc.cl

Abstract

When learning tasks over time, artificial neural networks suffer from a problem known as Catastrophic Forgetting (CF). This happens when the weights of a network are overwritten during the training of a new task causing forgetting of old information. To address this issue, we propose MetA Reusable Knowledge or MARK, a new method that fosters weight reusability instead of overwriting when learning a new task. Specifically, MARK keeps a set of shared weights among tasks. We envision these shared weights as a common Knowledge Base (KB) that is not only used to learn new tasks, but also enriched with new knowledge as the model learns new tasks. Key components behind MARK are two-fold. On the one hand, a metalearning approach provides the key mechanism to incrementally enrich the KB with new knowledge and to foster weight reusability among tasks. On the other hand, a set of trainable masks provides the key mechanism to selectively choose from the KB relevant weights to solve each task. By using MARK, we achieve state of the art results in several popular benchmarks, surpassing the best performing methods in terms of average accuracy by over 10% on the 20-Split-MiniImageNet dataset, while achieving almost zero forgetfulness using 55% of the number of parameters. Furthermore, an ablation study provides evidence that, indeed, MARK is learning reusable knowledge that is selectively used by each task.

1 Introduction

Among the cognitive abilities of humans, memory is one of the most relevant. However, memories are fragile. In particular, studies from cognitive psychology show that memories can be lost when new knowledge disrupts old information, in a process known as task interference [1, 2]. In the case of artificial neural networks (ANNs), task interference is behind the problem known as Catastrophic Forgetting (CF) [3, 4, 5, 6, 7]. In effect, as a model learns a new task, its weights are overwritten causing catastrophic forgetting of old information.

Unfortunately, in the case of ANNs, task interference is more severe than in the case of humans. This is evident for the case of continual learning, *i.e.*, the case when data from new tasks is presented sequentially to the learner, who does not have access to previous data. Under this training scheme, when a previously trained model is retrained on a new task, it usually suffers a significant drop in performance in the original task [8, 9, 10]. Previous works to prevent CF have followed two main strategies. The first strategy [10, 8, 11, 12, 13] consists of avoiding the modification of parameters

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

that are key to solve previous tasks. Specifically, when facing a new task, a regularization term ensures that critical parameters are modified as little as possible. In general, this approach shows satisfactory performance in problems that involve few tasks, however, when the number of tasks increases, problems such as accumulated drifting in weight values and interference among them, make this approach difficult to scale. The second strategy [14, 15, 16, 17, 7, 18] consists of introducing structural changes to the architecture of a model. This includes methods that reserve part of the network capacity to learn each task [9], and methods that use special memory units to recall critical training examples from previous tasks [17, 19]. The main problems with these methods are the extra model complexity and the need of an efficient method to recall key information from previous tasks.

In contrast to these previous strategies, when learning new tasks, humans continuously associate previous experience to new situations, strengthening previous memories which helps to mitigate the CF problem [20]. Taking inspiration from this mechanism, we propose MetA Reusable Knowledge or MARK, a new model based on a learning strategy that, instead of mitigating weight overwriting or learning independent weights for different tasks, uses a metalearning approach to foster weight reusability among tasks. In particular, we envision these shared weights as a common Knowledge Base (KB) that is not only used to learn new tasks, but also enriched with new knowledge as the model learns new tasks. In this sense, the KB behind MARK is **not given by an external memory that encodes information in its vectors**, but by a *trainable model that encodes shared information in its weights*. As a complementary mechanism to query this KB, MARK also includes a set of trainable masks that are in charge of implementing a selective addressing scheme to query the KB.

Consequently, to build and query its shared KB, MARK exploits two complementary learning strategies. On the one hand, a metalearning technique provides the key mechanism to meet two goals: i) encourage weight updates that are useful for multiple tasks, and ii) enrich the KB with new knowledge as the model learns new tasks. On the other hand, a set of trainable masks provides the key mechanism to selectively choose from the KB relevant weights to solve each task.

In terms of its inner operation, MARK works by first forcing the model to reuse current knowledge via functions that detect the importance of each pattern learned in the KB, and later expanding its knowledge if past knowledge is not sufficient to successfully perform a task. As an intuition behind how MARK works (not to be taken literally), when facing different tasks related to the classification of animals, MARK may take advantage of its metalearning approach to learn relevant features to discriminative concepts such as fur, paws, fin, and ears. Afterwards, when solving a task related to the discrimination of horses vs zebras, it might use a mask to focus on features related to fur rather than fins or ears.

We perform extensive experiments to analyze the inner workings behind MARK. In particular, we demonstrate that, indeed, MARK is able to accumulate knowledge incrementally as it learns new tasks. Furthermore, in terms of the pieces of knowledge used to solve each task, our experiments confirm that the use of task dependent feature masks results critical to the success of the method. We summarize our contributions as follows:

- We propose MARK, a novel method for continual learning scenarios that is based on two complementary mechanisms: i) A new method to build a KB that incrementally accumulates relevant knowledge from several tasks while avoiding CF problems, ii) A mechanism to query this KB to extract relevant knowledge to solve new tasks.
- An extensive experimental evaluation and public implementation of MARK that demonstrate its advantages with respect to previous works.

2 Continual Learning Scenario

Following previous work on continual learning [21], we consider a task incremental scenario, where task identity is provided at training time. Each task t consists of a new data distribution $D^t = (X^t, Y^t, T^t)$, where X^t denotes the input instances, Y^t denotes the instance labels, and T^t is a task ID. This task ID is required during training and inference. The goal is to train a classification model $f : X \longrightarrow Y$ using data from a sequence of T tasks: $D = \{D^1, ..., D^T\}$. Following the usual setup for continual learning [21], each task is presented sequentially to the model. Our scenario allows for unlimited use of data from the current task, but after switching to a new task, this data is no longer available. Furthermore, there is no intersection between the classes of different tasks, but they share a similar domain. This domain intersection is key to take advantage of common patterns among

the tasks. Next section presents the details behind MARK, our proposed model for incremental learning in the previous scenario.

3 Method

When learning tasks sequentially, humans build upon previous knowledge, leading to incremental learning. In contrast, in a similar scenario, ANNs devote all of their resources to the current task, leading to the problem of CF. Taking inspiration from the behavior of humans, an appealing idea is to provide the model with a KB that, as the model faces new tasks, incrementally captures relevant knowledge. Using this shared KB, the model can associate previous experience to new situations, mitigating the CF problem. To implement this idea, we have to address two main challenges: i) How do we build this KB incrementally?, and ii) How do we query this KB to access relevant pieces of knowledge?

To address the first challenge, we leverage metalearning in order to train a KB from data. Specifically, we use a metalearning strategy known as episodic training [22]. This strategy consists of sampling from a task pairs of support and query sets to simulate training data from a large number of mini-tasks. These mini-tasks are then used to bias the learner to improve its ability to generalize to future tasks. In turn, this generalization leads to our goal: to capture relevant knowledge that can be reused to face new tasks.

To address the second challenge, on top of the KB, we train mask-generating functions for each task. These masks provide a suitable mechanism to selectively access the information encoded in the weights of the KB. We can envision these masks as a query that is used to access the intermediate activations of the KB. Following [23], these masks depend solely on each specific task and input. In this way, given an input, MARK uses the corresponding mask to query the KB generating a feature vector. This vector is then used by a task dependent classification head to output a prediction. We describe next the details behind this operation.

3.1 Model Architecture

Figure 1 shows a schematic view of the operation behind MARK, where the main modules are:

- **Feature Extractor (F^t)**: This module is in charge of providing an initial embedding for each input X_i, i.e., F^t takes input X_i and outputs a vector representation F_i^t. In our case, we test our method using visual recognition applications, therefore, F^t is given by a convolutional model. It is important to note that model F^t can be shared among tasks or it can be specific to each task.
- **Knowledge Base (KB):** This is the main module behind MARK. It is in charge of accumulating relevant knowledge as the model faces new tasks. In our implementation, we use a convolutional architecture with B blocks. This part of the model is shared across tasks.
- **Mask-Generating functions (M^t):** These modules take as an input feature vector F_i^t and produce an instance and task-dependent mask M_i^t for each block of the KB. Each mask consists of a set of scalars, one for each channel in the convolutional blocks of the KB, that multiply the activation of each channel. These masks are critical to select what knowledge is relevant to each instance and task. In our implementation, we use fully connected layers.
- **Classifier (C^t):** These modules correspond to task-dependent classification heads. Its input $F_{i,KB}^t$ is given by a flattening operation over the output of the last block of the KB. Given the task ID of an input X_i, the corresponding head outputs the model prediction. In our implementation, we use fully connected layers.

Specific details can be found in Section A.1. As shown in Figure 1, the flow of information in MARK is as follows. Input X_i goes into F^t to extract the representation F_i^t. This representation is then used by M^t to produce the set of masks that condition each of the blocks in the KB. The same input X_i enters the mask-conditioned KB leading to vector $F_{i,KB}^t$ used by the classification head. Finally, classifier C^t generates the model prediction, where t is the task ID associated to input X_i.

3.2 MARK Training

Algorithm 1 describes the training process behind MARK. The first step consists of initializing the KB by training it end-to-end on the first task, without using metalearning and mask functions. In

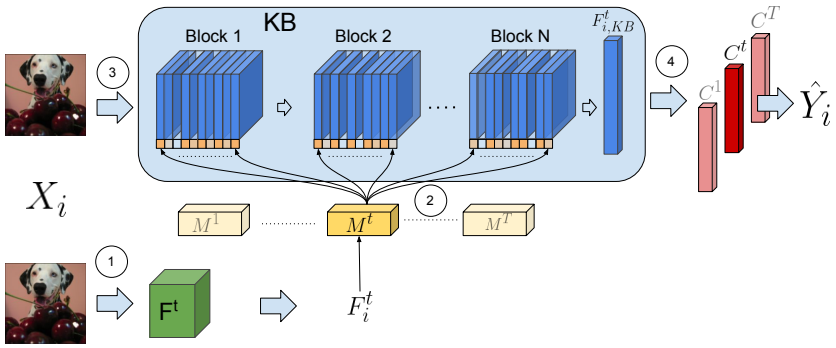


Figure 1: Given an input X_i from task t, (1) We use a feature extractor F^t to obtain F^t_i. (2) F^t_i is then passed to mask function M^t to generate mask M^t_i. Afterwards, (3) the same input X_i enters the KB, which has intermediate activations modulated by M^t_i. Finally, (4) the modulated features go through a task-dependent classifier C^t that performs the class prediction for X_i.

other words, we perform the KB initialization using the regular training procedure of a convolutional neural network for a classification task. After this, we train MARK sequentially on each task by alternating three main steps:

1. **KB Querying.** We train task-dependent mask-generating functions that are used to query the KB using vector F^t_i. Also, we concurrently train the task classifier for the current task. Notice that, leaving aside the KB initialization, in this step each new task is trained using only accumulated knowledge from previous tasks.
2. **KB Update**. We use a metalearning strategy to update the weights in the KB. This scheme allows fostering KB updates that favor the acquisition of knowledge that can be reused to face new tasks.
3. **KB Querying.** After updating the KB using knowledge from the current task, we repeat the querying process to finetune the mask-generating functions and task classifier using these new knowledge. Notice that during this step the KB is kept fixed.

The intuition behind the application of the three previous steps is as follows. We initially query the KB using the accumulated knowledge from previous tasks. This forces mask functions and classifiers to reuse the available knowledge. When that knowledge is exhausted, we proceed to add knowledge from the current task into the KB. Finally, we take advantage of this newly updated KB to obtain our final mask-functions and classifiers for a given task. We describe next the main steps behind the process to query and update the KB.

Algorithm 1: MARK - Training Process

Components:

- D^t: Dataset for task t.
- F^t: Feature extractor for task t.
- KB: Knowledge Base.
- M^t: Mask-generating function for task t.
- C^t: Classifier for task t.

$KB \leftarrow Train\left\{KB(D^1)\right\}$ */*Init KB */*
for *task* $t \leftarrow 1$ **to** T **do**
  */*1. Obtain feature vectors F^t_i*/*
  $F^t_i \leftarrow \left\{F^t(D^t_i)\right\}, i \in \{1, \ldots, |D^t|\}$
  */*2. Train M^t and C^t using current KB*/*
  $Train\{M^t \text{ \& } C^t\}$
  */*3. Update KB using metalearning*/*
  $KB \leftarrow$ KB-Update */*Algorithm 2*/*
  */*4. Finetune M^t and C^t with updated KB*/*
  $Train\{M^t \text{ \& } C^t\}$
end
Output: Trained modules KB, M^t, C^t.

Algorithm 2: KB-Update

Components:

- KB: Knowledge Base.
- C^k: Temporal Classifier for batch k
- E_{outer}: number of training updates for KB (outer loop)
- E_{inner}: number of inner loop epochs.

for *e in* E_{outer} **do**
  Generate K batches of data from D^t
  for k *in* K **do**
    $KB^k \leftarrow KB$ */*Copy of KB*/*
    $Initialize(C^k)$
    Train KB^k and C^k with batch k for E_{inner} epochs.
  end
  $\nabla KB \leftarrow \frac{1}{E_{inner}} \sum_k^K \gamma_k (KB^k - KB)$
  $KB \leftarrow KB - \alpha \nabla KB$
end
Output: KB */*output updated KB*/*

3.2.1 KB Querying

Once we obtain feature vectors through the use of feature extractor F^t, the model can learn which are the modules in the KB that can best solve the current task. In this training stage, the model trains functions to learn how to use the knowledge available in the KB, focusing solely on reusing knowledge from previous tasks, *without modifying the KB*. In particular, in this step we only train M^t and C^t. Both are trained end-to-end, while keeping the KB weights frozen.

As we generate masks for each intermediate activation of our model, strictly speaking we have a total of B mask-generating functions. However, to facilitate the notation, we subsume all such functions under the term M^t and think of its output as the concatenation of the results of those B functions. Eq. 1 shows the function M^t, where a mask M_i^t is obtained given an input X_i from task t. Our implementation of these functions consists of a linear function with parameters $W^{t,M}$, and an activation function ρ, which we implement as a ReLU.

$$M_i^t = M^t(F_i^t) = \rho((W^{t,M})^T F_i^t) \tag{1}$$

Masks generated in this process are encouraged to have two effects: first, to give a signal of how important a specific module from the KB is for the current input; second, to make sure that gradient updates are done where they really matter. If an activation map is irrelevant for a certain task, then the value of the corresponding mask will be zero, making the gradient update associated to that activation being zero as well.

3.2.2 KB Update

The purpose of this training step is to add new knowledge *from the current task* to the KB. As a further goal, we aim to enrich the KB with features that are highly general, i.e., they can be used to solve several tasks. To achieve this, we use metalearning as a way to force the model to capture knowledge that can be reused to face new tasks.

Figure 2 shows an schematic view of the metalearning procedure that we use to train MARK. This procedure consists of an adaptation of the training process presented in [24]. Specifically, we create a set of K mini-tasks, where each mini-task consists of randomly sampling from the current task a set of H classes and h training instances per class. This allows us to create a mini-task that differs from the main task, finding weights not specific to it. We train one copy of the model using each mini-task for E_{inner} epochs. We refer to the copy k trained for e epochs as KB_e^k. For each mini-task we use a temporary classifier C^k initialized with the parameters of C^t. We discard this classifier after the final iteration of the inner-loop.

Following [25], our method for episodic training consists of two nested loops, an inner and an outer loop. The inner loop is in charge of training the copies of our KB for the current mini-task while the outer loop is in charge of updating the KB weights following a gradient direction that leads to fast adaptation to new mini-tasks. During each inner loop, KB^k and C^k are trained end-to-end for E_{inner} epochs.

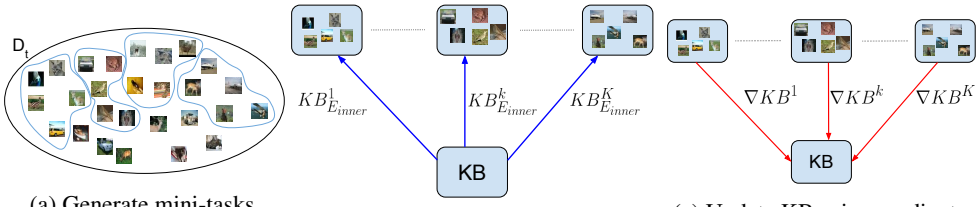
To simulate the outer-loop and update the KB, we follow Eq. 2. Specifically, for each k, we average the difference between the parameters of the KB before KB_0^k and after $KB_{E_{inner}}^k$ in the inner loop (see Eq. 2). This is simply an average of the sum of the cumulative gradients for each model KB^k. This is repeated E_{outer} times.

$$KB = KB - \alpha \nabla KB \qquad \nabla KB = \frac{1}{E_{inner}} \sum_k^K \gamma_k (KB_{E_{inner}}^k - KB_0^k) \tag{2}$$

The subtraction in Eq. 2 is weighted by γ_k. We compute γ_k as shown in Eq. 3, taking as a reference the accuracy of each model on a validation batch from the same task t.

$$\gamma_k = \frac{acc_k}{\sum_j^K acc_j} \tag{3}$$

By updating weights with a weighted average of solutions of all meta-tasks, we expect that the new values of the full model should be (on average) an appropriate solution for different tasks.



(a) Generate mini-tasks. (b) Train copy of KB in each mini-task. (c) Update KB using gradients.

Figure 2: KB update using metalearning. a) Given a task t, we randomly generate a set of K mini-tasks, where each mini-task consists of a subset of classes from the original task. b) For each mini-task, we train an independent copy of the current KB for a fixed amount of epochs, leading to K models. c) Afterwards, we calculate gradients with respect to the loss function of each of these models using a hold-out set of training examples. Finally, we update the KB using a weighted average of these gradients.

4 Experiments

For our experiments, we use two benchmarks used in previous works [18, 8, 26]. The first one is 20-Split CIFAR-100 [27] that consists of splitting CIFAR-100 into 20 tasks, each one with 5 classes. The second one is 20-Split MiniImagenet [22] that consists of dividing its 100 classes into 20 tasks.

In our experiments, we compare MARK with recent state of the art methods: Hard Attention to the Tasks (HAT) [7], A-GEM [28], Adversarial Continual Learning (ACL) [18], GPM [29], Experience Replay and SupSup[30]. We also include a Multitask baseline to serve as an upper bound for performance, where all tasks are trained jointly. In the Appendix, we include a comparison of the size of these alternative models with respect to MARK. All hyperparameters used for each method are also included in the Section A.2.

In terms of the feature embedding F^t used to represent input instances, we test the following approaches:

- **MARK-Task:** we train F^t for each task adding a classifier on top of it that is trained using D^t. After training F^t, this classifier is discarded.
- **MARK-Random:** F^t consists of a set of random weights. All tasks share the same F^t.
- **MARK-Resnet:** all tasks share a Resnet-18 pre-trained on Imagenet as a feature extractor.

We run all of our experiments using 3 different seeds. In terms of hyperparameters, we use SGD with a learning rate of 0.01 and a batch size of 128. Each task is trained for 50 epochs. To update the KB, we use 10 meta-tasks (K), trained for E_{inner} =40 epochs, each with a learning rate of 0.001. We repeat this training stage 20 times (E_{outer}). Code is released at: `https://github.com/JuliousHurtado/meta-training-setup`.

4.1 Metrics

To quantify the performance of our method, we use two metrics: average accuracy (Acc) and *backward transfer* (BWT). These are given by:

$$Acc = \frac{1}{T}\sum_{i=1}^{T} Acc_{T,i} \qquad BWT = \frac{1}{T-1}\sum_{i=1}^{T-1} Acc_{T,i} - Acc_{i,i} \tag{4}$$

Acc measures average performance over the T tasks after the sequential learning. BWT measures how much performance is lost on previous tasks after sequential learning. As a measure of efficiency, we also consider the amount of memory used by each method, considering number of parameters and extra temporal information needed by each method.

4.2 Results

We first analyze the general performance of our method with respect to the alternative approaches. We highlight the version of MARK where we pre-train F^t using data from each task (MARK-Task).

Table 1: Results using 20-split CIFAR-100 and 20-split MiniImagenet datasets. Standard deviation for 3 runs is listed in parentheses.

	CIFAR100			Mini-Imagenet		
Method	Acc (std.)%	BWT%	Mem%	Acc (std.)%	BWT%	Mem%
HAT	76.96 (± 1.2)	**0.01**	146%	59.45 (±0.1)	-0.04	198%
A-GEM	61.88 (± 0.2)	-16.97	205%	52.43 (±3.1)	-15.23	165%
ACL	78.08 (± 1.3)	0	134%	62.07 (±0.5)	0	195%
GPM	76.67 (± 3.1)	-0.42	66%	60.41 (±0.6)	0	70%
Exp. Replay	65.90 (± 1.2)	-16.9	130%	-	-	-
SupSup	77.58 (± 1.3)	0	26/111%[1]	-	-	-
Multitask	84.08 (± 1.2)	0	159%	74.44 (±1.6)	0	99%
MARK-Random	60.91 (± 0.4)	-1.73	41%	33.02 (±0.7)	**1.29**	43%
MARK-Task	**78.31** (± 0.3)	-0.27	100%	**69.43** (±1.6)	-0.39	100%
MARK-Resnet	86.29 (± 0.1)	-3.05	345%	93.55 (±0.2)	-2.52	126%

While Table 2 shows that MARK-Resnet leads to better results, it uses additional information due to its pre-training using ImageNet. It is important to note that embeddings F_i^t are not directly used to classify the corresponding instances. Instead, they are applied to generate the masks used to query the KB. Aside from its impact on accuracy, we observe no meaningful impact of the input representation on BWT.

As shown in Table 2, for both datasets, MARK-Task outperforms all competing methods in terms of average accuracy, while showing no sign of CF (BWT is close to zero). This is especially remarkable for Mini-Imagenet, as competing methods use complex AlexNet-based architectures during training versus our simple convolutional model. It is also noteworthy that MARK-Task accomplishes this while using almost half less memory than its closest competitor, ACL. This is because MARK-Task reuses the same stored KB for all tasks, while for each task it needs to store a small mask-generating function and classifier. Other methods either need to store extra parameters for adversarial training [18] or they require access to past gradients [28][29].

4.2.1 Evolution of Weight Updates

We hypothesize that the positive results achieved by MARK are due to the knowledge stored in its KB being highly reusable. To test this, we analyze the updated weight during training of all tasks. As a baseline, we consider a version of MARK that does not include metalearning and mask-filtering steps. In this experiment, weights are considered updated if their average deviation is over a certain threshold after training a task. Please see the appendix for specific details.

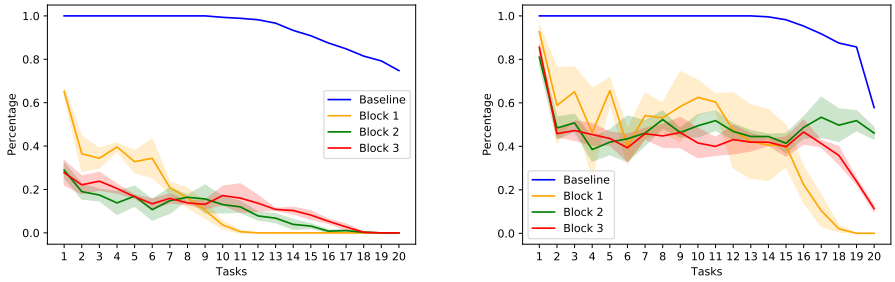
Figure 3 shows the percentage of weights that change after training each task. We observe two distinct phenomena: 1) MARK changes a rather small number of weights compared to the baseline, and 2) As more tasks are trained, the amount of weight updates dwindles.

We attribute both phenomena to MARK's ability to reuse previous knowledge to tackle new tasks. Thus updates should be needed mostly when learning new knowledge. The dwindling amount of updates can be linked to MARK's ability to develop its KB incrementally, thus new tasks are far more likely to be tackled with knowledge already in the KB. We believe that task similarity plays a crucial part in this behaviour, with each new task learned facilitating MARK's ability to find similarities for future tasks. However, to elucidate how important is task similarity to achieve these results is beyond the scope of this study. We leave this as an interesting avenue for future work.

4.2.2 Task Learning Speed

We hypothesize that if the KB is storing reusable knowledge, then learning speed for new tasks should increase as the KB incrementally contains more knowledge. We analyze the accuracy curves for each task when using MARK versus a model that is sequentially trained without including any mechanism

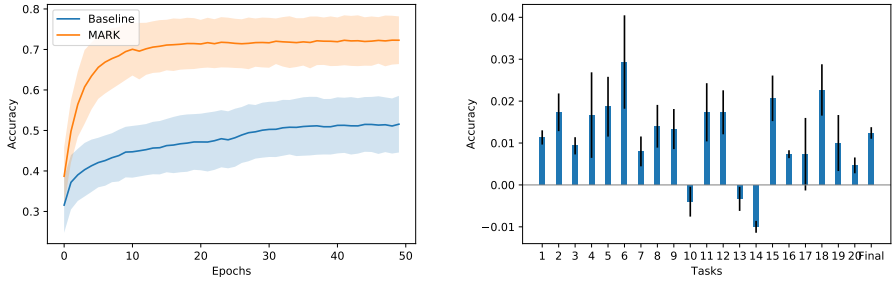
[1]Memory usage of SupSup depends on whether the model is generated from a seed or stored fully.



(a) 20-Split CIFAR-100 (b) 20-Split MiniImageNet

Figure 3: Percentage of updated weights in different blocks of the KB during sequential training. As a baseline, we consider a version of MARK that do not include metalearning and mask-filtering. Using MARK fewer updates are required, suggesting that new tasks add less knowledge to the KB due to incremental learning.

to avoid CF. Given that MARK trains its masks and classifiers two times per task, to be fair, we train the baseline for twice as many epochs as MARK. Consequently, we report results for the baseline every two epochs. Figure 4a shows the comparison in speed between the two models. We observe that indeed, on average, MARK achieves higher accuracy and stabilizes quicker than the baseline. This provides further evidence about the ability of MARK to encode reusable knowledge.



(a) Training Speed (b) Accuracy Gain

Figure 4: (a) Average test accuracy in the CIFAR-100 dataset achieved by MARK and a baseline model that is sequentially trained without including any mechanism to avoid CF. MARK achieves both greater accuracy and faster stabilization. (b) Accuracy in a task when we re-training the task after the model complete its sequential learning using all available tasks. We observe an increase in performance for most tasks, suggesting successful knowledge accumulation in the KB.

4.2.3 Importance of Metalearning and Mask Functions

We study the impact of the metalearning strategy used to update the KB and the mask-generating functions used to query the KB. To do this, we compare MARK against three ablations:

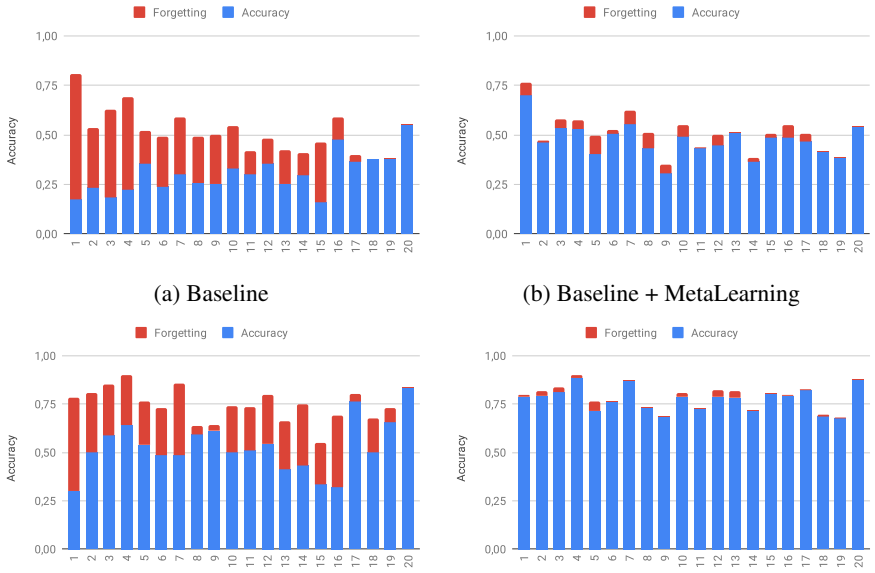
- **Baseline:** Simple sequential learning with no metalearning or mask-generating functions. We use the same architecture as the KB.
- **Baseline + ML:** We improve the baseline by adding metalearning, i.e., KB update.
- **Baseline + Mask:** We improve the baseline by adding task-specific mask functions.

Figure 5 shows that the baseline suffers from both significant forgetting and reduced performance. In contrast, when we include metalearning (*Baseline + ML*) forgetting is reduced. Similarly, when we only add mask-functions (*Baseline + Mask*), there is a boost in performance but forgetting also increases. By using metalearning and mask-functions, our full MARK-Task model achieves high accuracy with almost no forgetting.

Two important observations can be extracted from these experiments: 1) The forgetfulness of *Baseline + ML* is more significant than that obtained in MARK (-3% v/s -0.25%), showing that learning to use prior knowledge through masks can also help reduce forgetfulness. 2) The maximum accuracy averaged over tasks (without forgetting) is higher in MARK than in *Baseline + Mask* (78% v/s 74%), showing that training with metalearning helps knowledge transfer between tasks. This behaviour might seem counterintuitive as we metalearn only using the current task. However, an important part of MARK is forcing the reuse of previously useful features which biases solutions to start from a point which is useful for previous tasks. It is also worthwhile to note that it is known [24] that a REPTILE-style update (as is used in MARK) optimizes for synergistic gradient updates between minitasks. This might also help learn features that are more reusable, which might explain part of the success of MARK.

4.2.4 Incremental Construction of KB

We expect that, as MARK faces new tasks, its KB should be enriched with new knowledge. To test this hypothesis, we analyze the effect of training again each task after the model completes its sequential learning using all available tasks. As expected, Figure 4b shows that indeed, for most tasks, there is an absolute increase of 1.24% in performance when they are revisited after completing the sequential learning cycle. However, this increase might have to do with relearning what was forgotten rather than learning new knowledge. Thus, we also compared the difference between the maximum accuracy achieved during training for a task versus its final accuracy after retraining. We observed an absolute 0.97% average increase in accuracy as well, which shows that indeed new knowledge was used by earlier tasks.



(a) Baseline

(b) Baseline + MetaLearning

(c) Baseline + Masks

(d) Full model: MARK-Task

Figure 5: CF and accuracy for different versions of MARK-Task tested on 20-Split CIFAR100. (a) Without using metalearning and mask-functions, performance is low and CF is high. (b) Adding only metalearning, performance is still low, but there is almost no forgetting. (c) Adding only mask-functions, performance increases but forgetting is still high. (d) MARK-Task, our full model, achieves high performance with almost no forgetting.

5 Limitations

Our method relies on a base assumption that there is common structure to be learned between tasks. Thus, if this does not happen it will perform poorly (as any method which expects to reuse parameters for different tasks). Therefore, MARK is sensitive to the quality and structure of the first task.

As our method relies on task-specific modules, the amount of parameters scales linearly with the amount of tasks, possibly hindering its practical use in settings with a large amount of tasks.

It is not a straightforward task to extend our model to use memories for replay, since our focus has been on the scenario where we can't have access to past data after switching tasks. It is unclear where replay should happen, nor what should be stored as both masks and samples could work well.

6 Related Work

Previous works have tackled the problem of CF using different strategies. Memory-based methods mitigate CF inserting data from past tasks into the training process of new tasks, continuously re-training previous tasks [31], either with raw samples [32, 26], or minimizing gradient interference [33, 28]. Other works such as [34, 35] train generator functions (GANs) or autoencoders [36] to generate elements from past distributions. They seek memory-efficiency by generating examples. Similarly, [37, 38] seek to be memory-efficient by saving feature vectors of instances from previous tasks, while learning a transformation from the feature space of past tasks to current ones.

Other methods focus on limiting the flexibility of new tasks to modify the current model. The typical approach penalizes weight modifications or freezes a subset of the model. This can be achieved by adding weight regularizations [10, 8] or using masks to freeze parts of the model [16, 15]. Another alternative is to increase network capacity by adding extra parameters [9, 18]. Works similar to ours have proposed to find a path of relevant weights to solve the task [39], freezing used weights, and limiting learning of new tasks. Others use different functions as components in the network, either Hypernetworks [40], Deep Artificial Neurons (DANs) [41], or Compositional Structures [42], so that the network components are more flexible during learning.

Metalearning [43] is the idea of learning to learn. It has been used as a training strategy in sequence learning [44, 45] and continual learning scenarios [46, 47, 48, 49, 50]. In the case of [44, 45], the objective of meta-learning is to learn a starting point from where future sequences can learn sub-components of the model in the so-called inner-loop. In the case of MARK, we use the meta-learning strategy while learning new tasks, encouraging the reuse of the weights before and after the modification of the KB.

The idea of conditioning the weights of a model has being used before. [51] normalizes the output of a convolution layer by squeezing the activation maps via linear functions. [23] conditions the output of convolutional layers using the question in a VQA problem. However, as far as we know, this idea has not been combined with metalearning to update a KB.

7 Conclusions

In this work, we present MARK, a novel method for continual learning scenarios that is based on the construction and query of a KB that uses metalearning to incrementally accumulate relevant knowledge from different tasks. Our experiments indicate that the use of metalearning to build the KB is the crucial factor to mitigate CF, while the use of mask-functions to query the KB is the key factor to achieve high performance. Specifically, MARK achieves state of the art results in both 20-Split CIFAR-100 and 20-Split MiniImageNet, while suffering almost no BWT. Compared to ACL, the best performing alternative method, MARK uses 55% of the parameters, 51% of the memory requirements, while achieving almost a 12% increase in average accuracy on the 20-Split MiniImageNet benchmark.

Acknowledgments and Disclosure of Funding

This work was supported by National Agency of Research and Development (ANID) via National Doctorate Scholarships. This project has also received funding from the Millennium Institute of Foundational Research on Data (IMFD).

References

[1] Alan D Baddeley. *Human memory: Theory and practice*. Psychology Press, 1997.

[2] Saul A McLeod. Forgetting. https:www.simplypsychology.orgforgetting.html, 2008. [Online; accessed 30-Sept-2019].

[3] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, 2017.

[4] Nicolas Y Masse, Gregory D Grant, and David J Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 2018.

[5] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems*, 2017.

[6] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *IEEE International Conference on Computer Vision*, 2017.

[7] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, 2018.

[8] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, 2017.

[9] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[10] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017.

[11] Rahaf Aljundi, Marcus Rohrbach, and Tinne Tuytelaars. Selfless sequential learning. In *International Conference on Learning Representations*, 2019.

[12] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision*, 2018.

[13] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *The IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[14] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[15] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[16] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision*, 2018.

[17] Wenpeng Hu, Zhou Lin, Bing Liu, Chongyang Tao, Zhengwei Tao, Jinwen Ma, Dongyan Zhao, and Rui Yan. Overcoming catastrophic forgetting via model adaptation. In *International Conference on Learning Representations*, 2019.

[18] Sayna Ebrahimi, Franziska Meier, Roberto Calandra, Trevor Darrell, and Marcus Rohrbach. Adversarial continual learning. *Proceedings of the European Conference on Computer Vision*, 2020.

[19] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, , and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations*, 2019.

[20] Jeffrey D. Karpicke. *A powerful way to improve learning and memory*, 2016 (accessed June, 2020).

[21] Gido M van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[22] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 2016.

[23] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *Association for the Advance of Artifical Intelligence*, 2018.

[24] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[25] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

[26] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. *International Conference on Machine Learning*, 2019.

[27] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *http://www.cs.toronto.edu/ kriz/cifar.html*, 2009.

[28] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations*, 2018.

[29] Gobinda Saha and Kaushik Roy. Gradient projection memory for continual learning. In *International Conference on Learning Representations*, 2021.

[30] Mitchell Wortsman, Vivek Ramanujan, Rosanne Liu, Aniruddha Kembhavi, Mohammad Rastegari, Jason Yosinski, and Ali Farhadi. Supermasks in superposition. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15173–15184. Curran Associates, Inc., 2020.

[31] Sayna Ebrahimi, Suzanne Petryk, Akash Gokul, William Gan, Joseph E. Gonzalez, Marcus Rohrbach, and trevor darrell. Remembering for the right reasons: Explanations reduce catastrophic forgetting. In *International Conference on Learning Representations*, 2021.

[32] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017.

[33] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, 2017.

[34] Timothée Lesort, Hugo Caselles-Dupré, Michael Garcia-Ortiz, Andrei Stoian, and David Filliat. Generative models from the perspective of continual learning. In *International Joint Conference on Neural Networks*. IEEE, 2019.

[35] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in neural information processing systems*, 2017.

[36] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations*, 2018.

[37] Ahmet Iscen, Jeffrey Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. In *Proceedings of the European Conference on Computer Vision*. Springer, 2020.

[38] Lucas Caccia, Eugene Belilovsky, Massimo Caccia, and Joelle Pineau. Online learned continual compression with adaptive quantization modules. In *International Conference on Machine Learning*, pages 1240–1250. PMLR, 2020.

[39] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[40] Johannes von Oswald, Christian Henning, João Sacramento, and Benjamin F. Grewe. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020.

[41] Blake Camp, Jaya Krishna Mandivarapu, and Rolando Estrada. Continual learning with deep artificial neurons. *arXiv preprint arXiv:2011.07035*, 2020.

[42] Jorge A Mendez and Eric Eaton. Lifelong learning of compositional structures. In *International Conference on Learning Representations*, 2021.

[43] Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*. Springer, 1998.

[44] Khurram Javed and Martha White. Meta-learning representations for continual learning. In *Advances in Neural Information Processing Systems*, 2019.

[45] Shawn Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O Stanley, Jeff Clune, and Nick Cheney. Learning to continually learn. *European Conference on Artificial Intelligence*, 2020.

[46] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Page-Caccia, Issam Hadj Laradji, Irina Rish, Alexandre Lacoste, David Vázquez, et al. Online fast adaptation and knowledge accumulation (osaka): a new approach to continual learning. *Advances in Neural Information Processing Systems*, 2020.

[47] Joseph K J and Vineeth N Balasubramanian. Meta-consolidation for continual learning. In *Advances in Neural Information Processing Systems*, 2020.

[48] Jathushan Rajasegaran, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, and Mubarak Shah. itaml: An incremental task-agnostic meta-learning approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[49] Xu He, Jakub Sygnowski, Alexandre Galashov, Andrei A Rusu, Yee Whye Teh, and Razvan Pascanu. Task agnostic continual learning via meta learning. *arXiv preprint arXiv:1906.05201*, 2019.

[50] Matthias De Lange and Tinne Tuytelaars. Continual prototype evolution: Learning online from non-stationary data streams. *arXiv preprint arXiv:2009.00919*, 2020.

[51] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[52] Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido van de Ven, Martin Mundt, Qi She, Keiland Cooper, Jeremy Forest, Eden Belouadah, Simone Calderara, German I. Parisi, Fabio Cuzzolin, Andreas Tolias, Simone Scardapane, Luca Antiga, Subutai Amhad, Adrian Popescu, Christopher Kanan, Joost van de Weijer, Tinne Tuytelaars, Davide Bacciu, and Davide Maltoni. Avalanche: an end-to-end library for continual learning, 2021.

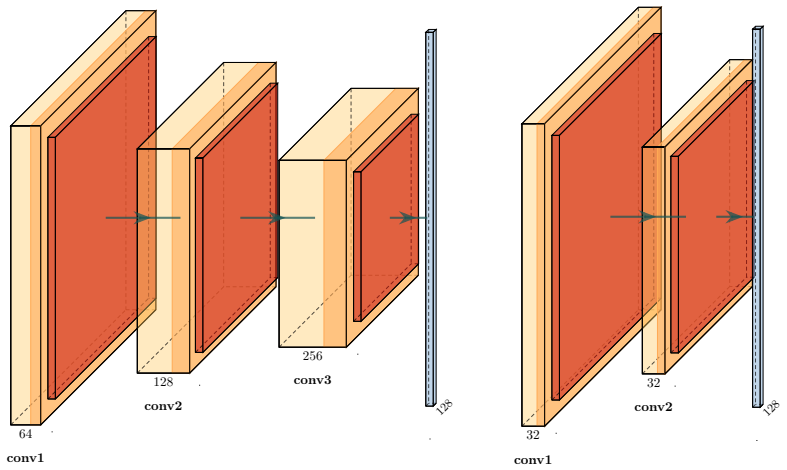
[53] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

A Appendix

A.1 Architecture

In this section, we review in detail the architectural choices we made for this work. As we mentioned previously, we impose no restriction on which models we can use on each part of MARK. Please, see Figure 6 for an schematic diagram of the architectures we used. In our case, for each component:

- **Feature Extractor (F^t)**: As we explained in the main document, this feature extractor can be any model, pretrained or otherwise. In the case of MARK-Task and MARK-Random, we use an architecture similar to the one used in the private model in [18], consisting of 2 convolutional layers with an activation function, batch normalization, and Max-Pooling, followed by a fully connected layer with ReLU. The number of filters in each layer depends on the dataset used. For CIFAR-100 we use 32 filters per block, while for MiniImageNet we use 8 filters per block. These numbers are similar as those used by ACL.
- **Knowledge Base (KB):** In this case, we use 3 shared blocks, each composed of 1 convolutional layer with 64, 128, and 256 components respectively, accompanied by an activation function (ReLU) and Max-Pooling. After these blocks, we add a fully connected layer that creates a representation to be input into the task classifier.
- **Mask-Generating functions (M^t):** As explained in the main document, we use a fully connected layer for each function accompanied by an activation function (ReLU). Input dimensions depend on the dataset, output is $(64 + 128 + 256) = 448$.
- **Classifier (C^t):** Is a fully connected layer.



(a) Knowledge Base

(b) Feature Extractor, F^t

Figure 6: Schematic representation of the architectures used in MARK.

A.2 Experiment Details

We run all of our experiments using 3 different seeds and implemented using PyTorch. In terms of hyperparameters, we use SGD with a learning rate of 0.01, weight decay of 0.01, a momentum of 0.9 and a batch size of 128. We set the momentum to 0 when updating the KB.

The F^t are trained for 50 epochs each when using MARK-Task. During the KB querying phase, mask functions and classifiers are trained for 50 epochs. In the KB Update step, we use 10 meta-tasks (K), trained for E_{inner} =40 epochs, each with a learning rate of 0.001. We repeat this training stage 15 times (E_{outer}). Our code will be made publicly available.

We also list the entire set of hyperparameters used for all methods which are compared against MARK. We use no learning rate scheduling for any of these experiments.

Table 2: Hyperparameters used for all methods compared against MARK

Method	Dataset	Learning Rate	Epochs	Optimizer	Batch Size	Result Origin
HAT	CIFAR100	0.01	As ACL	SGD	128	ACL Paper
A-GEM	CIFAR100	0.01	100	SGD	128	Avalanche[52]
ACL	CIFAR100	0.01	200	SGD	128	ACL Paper
GPM	CIFAR100	0.01	100	SGD	128	Paper Code
Exp. Replay	CIFAR100	0.01	100	SGD	128	Avalanche[52]
SupSup	CIFAR100	0.001	100	SGD	128	Paper Code
Multitask	CIFAR100	0.01	100	SGD	128	Code
HAT	MiniImageNet	0.01	As ACL	SGD	128	ACL Paper
A-GEM	MiniImageNet	0.01	100	SGD	128	ACL Paper
ACL	MiniImageNet	0.01	200	SGD	128	ACL Paper
GPM	MiniImageNet	-	-	-	-	GPM Paper
Multitask	MiniImageNet	0.01	100	SGD	128	Paper Code

A.3 Some notes for particular methods

- **A-GEM:** though A-GEM is originally run as a single pass algorithm, both on the ACL paper and on our own experiments ran it for multiple epochs to make results comparable. On our implementation for CIFAR100 we used a memory size of 200 per task.
- **Experience Replay:** We used a memory of 250 elements per task. This number is taken to make sure memory requirements between this baseline and MARK are similar.
- **SupSup:** We tried different values of sparsity until we were able to replicate the same results of their paper with their setup. Using that sparsity value, we replaced their model with a model similar to our KB but with extra parameters, similar to the model used in ACL and GPM. We tried to use 0.01 as learning rate, but results were much worse.
- **GPM:** For MiniImageNet, we were unable to find hyperparameters used either on the paper or code base.

A.4 CO2 Emission Related to Experiments

Experiments were conducted using a private infrastructure, which has a carbon efficiency of 0.432 $kgCO_2eq/kWh$. A cumulative of 75.5 hours of computation was performed on hardware of type GTX 1080 Ti (TDP of 250W).

Total emissions are estimated to be 8.16 $kgCO_2eq$ of which 0% were directly offset.

We run 3 experiments on a single GPU at the same time. Experiments on CIFAR-100 take 133 minutes, while on MiniImageNet take 622 minutes. We consider 6 variations for both datasets, each run with 3 different seeds.

Estimations were conducted using the MachineLearning Impact calculator presented in [53].

A.5 Parameters

Table 3 reports number of parameters for models used by different methods. Numbers for HAT, A-GEM and ACL are taken from the respective articles. GPM numbers were calculated using the author's code.

A.6 Importance of Metalearning and Mask Functions

We supply an augmented version of Figure 5 in Figure 7 which includes data from MARK-Random and MARK-Resnet. The figure shows that using different input representations for the mask-generating functions has direct impact on the method's accuracy, but has little to no impact on BWT.

We hypothesize this behaviour is taking place because MARK's masks are acting as a kind of self-attention mechanism. With richer features, this mechanism is able to select more appropriate information to solve any given task. However, as expected, richer features aid nothing in attenuating

Table 3: Model Parameters for different methods in the Catastrophic Forgetting literature. HAT, A-GEM, ACL numbers were taken directly from the respective articles. GPM, SupSup were calculated based on author's code. Experience Replay was calculated based on our own code.

Method	CIFAR100	Mini-Imagenet
HAT	6.8M	30.9M
A-GEM	6.4M	25.7M
ACL	6.3M	28.3M
GPM	2.4M	1.5M
SupSup	20.9M	-
Experience Replay	2.9M	-
Multitask	7.4M	15.4M
MARK-Random	1.7M	6.7M
MARK-Task	4.7M	15.6M
MARK-Resnet	16.1M	19.7M

Catastrophic Forgetting. The role of features is to provide information with which to decide, while Backward Transfer is about forgetting how to decide.

A.7 Critical Dimensions

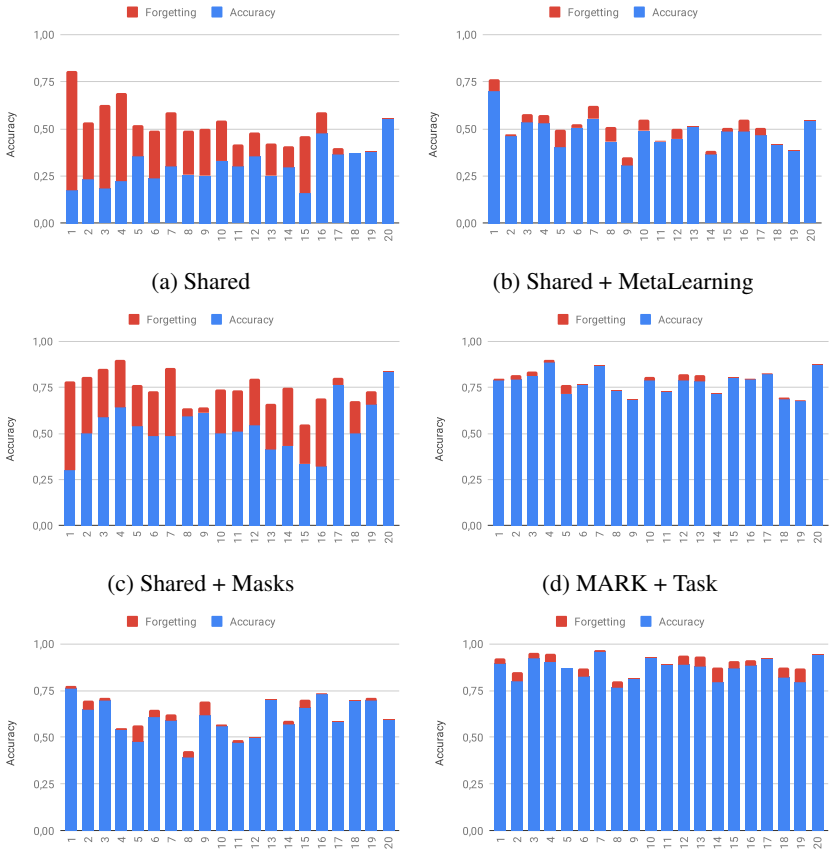
We also wanted to understand the importance of each dimension the masks were modulating. To test this, we take one of our trained models and selectively turn off (i.e. set to zero) the mask value associated with each activation map from the KB. We then analyze how many samples from the test set are incorrectly labelled compared to the regular model. We use this measure as a proxy for understanding the relevance of each activation map.

Table 4 shows that, for a given task, relatively few feature maps (on their own) have a critical impact on classification. However, a few feature maps show a disproportionate effect on classifying samples. These are feature maps that when turned off result in accuracy loss of 1% or more in task accuracy. We term these **critical dimensions**. Blocks closer to the output show an increase in the number of samples affected by a critical dimension. We hypothesize this is because layers closer to the output are believed to be more discriminative in nature, which would naturally lead to having more critical dimensions. This type of analysis, however, is limited in nature. We believe the impact of these dimensions should be studied not in isolation, but by their interactions with other dimensions. We have not found any tractable way of approaching such an analysis.

Table 4: The average percentage of dimensions that have a certain level of impact on performance for a model trained in 20-split CIFAR100. Results are averaged by the number of tasks. We see that most dimensions have no impact on samples, while a relative small part of them have relevant impact. Each column shows the percentage of dimensions that negatively impacts the accuracy by an X or greater percent for some task

	No impact	Impact	Critical						
Block	0%	>0%	>1%	>2%	>3%	>4%	>5%	>8%	>10%
Block 1	90.31	9.69	1.17	0.31	0.08	0.0	0.0	0.0	0.0
Block 2	95.0	5.0	1.52	1.25	1.09	0.98	0.86	0.78	0.66
Block 3	95.53	4.47	2.32	1.93	1.76	1.68	1.6	1.37	1.25

We then test how many of these critical dimensions are shared between tasks. We can see the results in Figure 8. A perfect KB would be composed of mostly reusable knowledge with some specific knowledge that is only relevant for a single task. What we see is that critical modules are shared between small groups of tasks (with one exception that is shared by 85% of the tasks). However, it is difficult to extrapolate conclusions over these findings. It is not clear how to quantify the ratio of specific knowledge to reusable knowledge, nor how to evaluate such a number if it was available.



(a) Shared

(b) Shared + MetaLearning

(c) Shared + Masks

(d) MARK + Task

(e) MARK + Random

(f) MARK + Resnet

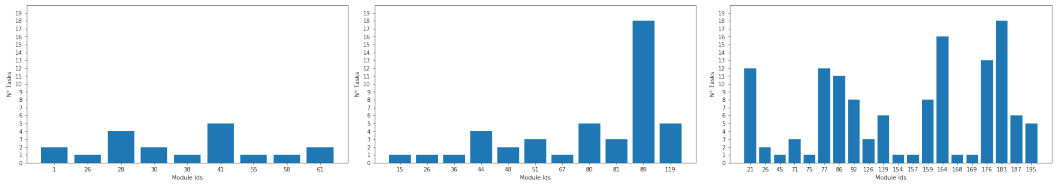
Figure 7: CF and accuracy for different versions of MARK tested on 20-Split CIFAR100. (a)Without using metalearning and mask-functions, performance is low and CF is high. (b) Adding only metalearning, performance is still low, but there is almost no forgetting. (c) Adding only mask-functions, performance increases but forgetting is still high. (d) MARK-Task, our full model,achieves high performance with almost no forgetting. (e) and (f) Quality of input features for mask-generating functions impacts overall accuracy but not forgetting. We hypothesize this is so because masks are acting as a self-attention mechanism that improves as richer features are provided.

A.8 Importance of Training Stages

We analyze the contribution of the different steps behind the training of MARK. As our method requires multiple steps in a particular order, it begs the question: are all of these steps necessary? To answer this, we compare two different versions of our model to MARK:

- **Feature:** This baseline is trained using only F^t with a classifier on top. It does not use KB, mask-functions M^t, or task specific classifiers C^t. This setup tests for no knowledge reuse between tasks.
- **No-Retraining:** We perform the KB-Querying phase of training only at the start of training a task. This setup tests the importance of retraining the mask functions and classifiers to take into consideration the new knowledge provided by the metalearning process.

Figure 9 shows the impact of each training method on task accuracy. We observe that, as expected, as we use more of MARK's components results are better. What is interesting is the difference between No Retraining and MARK at the beginning and by the end of training. For earlier tasks, not being able to capitalize on the full knowledge of the KB hurts the current's task accuracy greatly. However, as we near the end of our training, the difference between using the fully trained KB vs not using it



(a) Block 1 (64 modules) (b) Block 2 (128 modules) (c) Block 3 (256 modules)

Figure 8: Number of tasks that use a given critical dimension for each convolutional layer for a model trained on 20-split CIFAR100. Critical dimensions are defined as those that when turned off affect the correct classification of samples over a threshold. As we move closer to the output, layers increase the amount of critical modules, as well as intertask dependency on dimensions. However, no dimensions are universally useful. This suggests that while knowledge is reusable, there is still task-specific knowledge being stored in the KB. We use a threshold of N=5 samples which is a 1% deviation in accuracy.

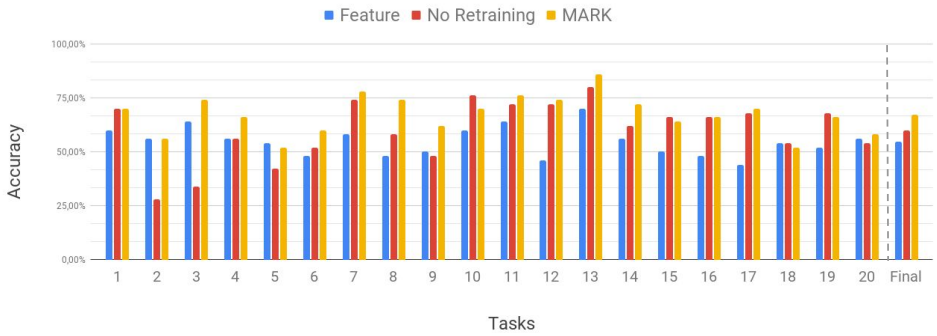


Figure 9: Accuracy in each of the 20 tasks in the 20-Split MiniImagenet sequence. First, we obtain the accuracy by only using the feature extractor. The No Retraining method shows a vast improvement on average, with MARK being the superior model. Note how the differences between No Retraning and MARK dwindle the more tasks we have trained. This suggests that MARK's KB is storing reusable knowledge that is available already at the start of training later tasks. The final three columns show the average of the complete sequence.

dwindles. We hypothesize this is because MARK's KB has already acquired sufficient knowledge from previous tasks, to perform well on the current task.

A.9 Number of Updated Parameters

In section 4.2.1 we mention a metric based on "number of updated parameters", however, we do not explain exactly what it means. We explain this now. For every task we compare the model before training on task t and after training. We compare the mean of the absolute difference between the *"before"* and *"after"* model against a threshold that is dependent on the standard deviation of the weights of the *"before"* model. If it's above this threshold we assume the weight was updated, else we don't. Standard deviations and means are calculated on a per-channel basis.

$$\sum_{i=1}^{n} \frac{|\theta^i_{before} - \theta^i_{after}|}{n} < \mu\sigma_{before}$$

NeurIPS Paper Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? Yes. For metalearning as key component for not forgetting, see section 4.2.3. For state of the art results, see Table 2. For reduced parameter use, see "Parameters" section in Appendix.
   (b) Did you describe the limitations of your work? Yes.
   (c) Did you discuss any potential negative societal impacts of your work? No. Our work is too broad and applicable for us to determine immediate applications that may be harmful to society.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? Yes.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? n/a
   (b) Did you include complete proofs of all theoretical results? n/a
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? Yes. We include a .zip file with our code, and we list some hyperparameters in the Experiments section and all of them in the appendix.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? Yes. See the appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? Yes. We include standard deviations on all of our figures. We run experiments with 3 seeds.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? Yes. See appendix for GPU details.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? Yes.
   (b) Did you mention the license of the assets? No. We use CIFAR-100 and MiniImageNet, very standard benchmarks.
   (c) Did you include any new assets either in the supplemental material or as a URL? No.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? No. We use CIFAR-100 and MiniImageNet, very standard benchmarks.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? No. We use CIFAR-100 and MiniImageNet, very standard benchmarks.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? n/a
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? n/a
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? n/a