Representation Memorization for Fast Learning New Knowledge without Forgetting

Fei Mi\textsuperscript{1*}, Tao Lin\textsuperscript{2,}, Boi Faltings\textsuperscript{2}

\textsuperscript{1}Huawei Noah’s Ark Lab,
\textsuperscript{2}Swiss Federal Institute of Technology in Lausanne (EPFL)
mifei2@huawei.com, \{tao.lin, boi.faltings\}@epfl.ch

Abstract

The ability to quickly learn new knowledge (e.g., new classes or data distributions) is a big step towards human-level intelligence. In this paper, we consider scenarios that require learning new classes or data distributions quickly and incrementally over time, as it often occurs in real-world dynamic environments. We propose “Memory-based Hebbian Parameter Adaptation” (Hebb) to tackle the two major challenges (i.e., catastrophic forgetting and sample efficiency) towards this goal in a unified framework. To mitigate catastrophic forgetting, Hebb augments a regular neural classifier with a continuously updated memory module to store representations of previous data. To improve sample efficiency, we propose a parameter adaptation method based on the well-known Hebbian theory Hebb [1949], which directly “wires” the output network’s parameters with similar representations retrieved from the memory. We empirically verify the superior performance of Hebb through extensive experiments on a wide range of learning tasks (image classification, language model) and learning scenarios (continual, incremental, online). We demonstrate that Hebb effectively mitigates catastrophic forgetting, and it indeed learns new knowledge better and faster than the current state-of-the-art.

1 Introduction

In real-life machine learning applications, new knowledge (e.g., new classes or data distributions) arrive gradually over time. The ability to quickly learn and accumulate new knowledge without forgetting old ones is a hallmark of artificial intelligence. Two major challenges prevent standard neural networks from being trained to achieve reasonable accuracy Wang \textit{et al.} [2020].

The catastrophic forgetting challenge is recently studied in the context of “continually/incrementally” learning a sequence of tasks. Various regularization-base methods Kirkpatrick \textit{et al.} [2017]; Rebuffi \textit{et al.} [2017]; Castro \textit{et al.} [2018] and memory-based approaches Grave \textit{et al.} [2017b]; Merity \textit{et al.} [2017]; Rebuffi \textit{et al.} [2017] have been proposed to mitigate catastrophic forgetting. The sample efficiency challenge is recently studied in the context of few-shot learning; a popular approach is “meta-learning” Finn \textit{et al.} [2017] that learns over a bunch of specifically structured meta-tasks.

However, existing methods often tackle these two challenges separately. To this end, we propose a method called Memory-based Hebbian Parameter Adaptation (Hebb) to tackle them in a unified framework. Hebb makes use of a memory component similar to Merity \textit{et al.} [2017]; Sprechmann \textit{et al.} [2018] that stores previous input representations to mitigate catastrophic forgetting. To improve sample efficiency on new knowledge, we propose a parameter adaptation procedure based on the well-known Hebbian theory Hebb [1949] during inference. It directly wires similar representations retrieved from the memory to the corresponding parameters of the classifier’s output network. The memory accessing operation and the parameter adaptation procedure can be easily computed such that Hebb can be easily plugged into different neural classifiers and learning scenarios.

Besides the standard continual learning setting Rebuffi \textit{et al.} [2017] to evaluate the ability to mitigate catastrophic forgetting, two other learning scenarios are considered to evaluate the ability to deal with the sample efficiency challenge. In the first incremental learning scenario, a pre-trained classifier is initially trained on a small dataset containing new knowledge. Then it is fixed to be evaluated w.r.t. future observations. In the second online adaptation scenario, we have no data on new knowledge initially, and the pre-trained classifier needs to continuously learn new knowledge through a single pass over new data in an online manner. These two learning scenarios are both practically critical. For example, the incremental learning scenario could simulate that a robot is shown some images of new objects before they appear in its routine tasks. In the online adaptation scenarios, a robot has to deal with new objects continuously.

Through extensive experiments on a wide range of learn-
ing tasks (image classification, language model) and learning scenarios (continual, incremental, online), we empirically demonstrate: (i) Hebb can be easily plugged into different neural classifiers and learning scenarios with trivial computation overhead. (ii) Hebb effectively mitigates catastrophic forgetting, indicated by its superior performance compared to MbPA Sprechmann et al. [2018] and EWC Kirkpatrick et al. [2017] in the continual learning setting. (iii) More importantly, Hebb notably improves sample efficiency for fast learning new classes and data distributions. It outperforms various state-of-the-art methods in both incremental learning and online adaptation scenarios, especially on new or infrequent classes.

2 Related Work

2.1 Memory-augmented Neural Networks

Recently, various memory modules (M) have been proposed to augment neural networks for remembering long-term information Graves et al. [2014]; Grave et al. [2017b,a]; Merity et al. [2017] or learning infrequent patterns Santoro et al. [2016]; Kaiser et al. [2017]; Sprechmann et al. [2018]; Mi and Faltings [2020].

There are many variants of how to read from M and mix the entries retrieved from M with the network computation. One approach is through some differentiable context-based lookup mechanisms Graves et al. [2014]; Santoro et al. [2016] for learning to match the current activation to past activations stored in M. However, these mechanisms often require strong memory supervision, and the size of the M has to be fixed. Another approach is using a simple mixture model. In this case, a non-parametric prediction is computed based on the similarity between the entries in M and the current input. The neural network’s prediction is directly interpolated with the non-parametric prediction from M. This approach has been shown simple but effective for language modeling Grave et al. [2017b,a], neural machine translation Tu et al. [2018], image classification Orhan [2018], and recommendation Mi and Faltings [2020]. Recently, Sprechmann et al. [2018] introduces MbPA to use nearest neighbors retrieved from M for parameter adaptation during model inference for the fast acquisition of new knowledge. MbPA++ de Mars in d'Autume et al. [2019] improves MbPA to better mitigate catastrophic forgetting through better memory management during training. The framework proposed in this paper is motivated by Sprechmann et al. [2018], and it mainly improves MbPA for better learning new knowledge.

2.2 Hebbian Learning

Hebbian theory Hebb [1949] is a neuroscientific theory attempting to explain “synaptic plasticity”, i.e., the adaptation of brain neurons during the learning process. For artificial neural networks, Hebbian theory describes a method of determining how to alter the weights between two neurons. It is also related to early ideas from psychology and neuroscience, called associative memory. In psychology, associative memory is the ability to learn and remember the relationship between unrelated items. In neuroscience, associative memory means that the information is stored by associative structures to bind representative patterns to their corresponding concepts or labels.

Recent approaches apply Hebbian theory to every single neuron connection for fast network weight learning. Ba et al. [2016] proposes a fast weight to augment the standard computation of RNNs. The fast weight is defined as the running average of the outer product of two hidden states in RNNs. It is multiplied to the current state and it is continuously updated to allow each new hidden state to be attracted to recent hidden states. Miconi et al. [2018] later augments the traditional connection between two neurons in general neural networks with a Hebbian trace. The Hebbian trace between two neurons is defined as a running average of the scalar product of the first neuron’s activation in the last timestamp and the second neuron’s activation in the current timestamp. The Hebbian trace is merged with the standard neuron connection through a differentiable plasticity optimized by SGD. The later extension Miconi et al. [2019] introduces a term parametrized by another neural network to learn how fast should new information be incorporated.

Instead of applying the Hebbian theory to every single neuron connection, we use it to directly wire the activation input to the output layer with the corresponding class label for fast binding new classes. Similar perspectives are recently proposed. For example, Munkhdalai and Trischler [2018] augments the layer preceding the Softmax layer with the Hebbian updates followed by a nonlinear activation for meta-learning. Rae et al. [2018] proposes a Hebbian Softmax layer during the normal model training phase to better learn infrequent vocabularies in language modeling tasks. The Hebbian update rule proposed in our paper is motivated by Rae et al. [2018], yet our Hebbian update rule is only applied to relevant entries retrieved from a continuously updated memory module for the purpose of fast learning new knowledge in an incremental or online manner.

3 Memory-based Hebbian Parameter Adaptation

This section introduces the Memory-based Hebbian Parameter Adaptation (MbPA) method to help standard neural classifiers mitigate catastrophic forgetting and improve sample efficiency. First, we introduce a memory component to store representations of past data. Then, we introduce the Hebbian update for fast learning new knowledge during inference, and we compare it with state-of-the-art (MbPA Sprechmann et al. [2018]). Lastly, we introduce a dynamic interpolation of the proposed Hebbian update and MbPA.

Background Neural classifiers can be visualized by two parts. The first part is a feature extractor \( y_0 \) to compute a input representation vector \( \mathbf{h}_x = y_0(x) \) for an input \( x \). The second part is an output network \( f_\omega \) for predicting \( \hat{y} = f_{\omega}(\mathbf{h}_x) \). A fully-connected layer with a Softmax activation is often used: \( f_{\omega}(\mathbf{h}_x) = \text{Softmax} (\mathbf{W}^T \mathbf{h}_x + \mathbf{b}) \). The weights \( \mathbf{W} \in \mathbb{R}^{d \times n} \) and the bias \( \mathbf{b} \in \mathbb{R}^{n} \), where \( d \) is the dimension of \( \mathbf{h}_x \) and \( n \) is the number of classes.
3.1 Memory Component

Motivated by Grave et al. [2017b]; Sprechmann et al. [2018], we design a memory module M in the form of key-value pairs, i.e., $M = \{(key, value)\}$, to tackle the catastrophic forgetting challenge. M is indexed by keys, and we define keys to be input representations while values are the corresponding class labels. Storing input representations rather than raw inputs also helps to preserve data privacy. Upon observing a training data $(x, y)$, we write a new entry to M by:

\[
\begin{align*}
\text{key} & \leftarrow h_x = g_0(x) \\
\text{value} & \leftarrow y
\end{align*}
\]  

(1)

To scale up to a large number of observations in practical scenarios, we utilize the FAISS library\(^1\) to implement a scalable retrieval method with Product Quantization Jégou et al. [2011] to achieve both computation and storage efficiency. Settings with limited and unrestricted memory sizes are both considered in later experiments.

To adapt the prediction for an input $x$ during inference, we retrieve a set of $K$ nearest neighbors of its representation $h_x = g_0(x)$ from M by:

\[
N = \{(h_k, y_k, c_k)\}_{k=1}^K
\]  

(2)

where $c_k$ is the closeness between $h_x$ and $h_k$, and we use the same kernel function $c_k = \exp(-|h_x - h_k|^2)$ as in Sprechmann et al. [2018]. Entries in N are used to adapt the parameters of $f_\omega$ and details are introduced next.

3.2 Hebbian Update

The general Hebbian theory Hebb [1949] is expressed as:

\[
W[i,j] = \frac{1}{n} \sum_{k=1}^{n} x_i^k x_j^k,
\]  

(3)

where $W[i,j]$ is the weight of the connection from neuron $i$ to neuron $j$; $x_i^k$ is the $k$-th input to the neuron $i$, and similarly for $x_j^k$; $k \in \{1...n\}$ and $n$ is the number of training samples. Therefore, the multiplication of $x_i^k$ and $x_j^k$ summing over $n$ training examples gives the weight $W[i,j]$ between the neuron $i$ and $j$. The intuition is: if nodes $i$ and $j$ are often activated together, they have a strong connection weight.

Next, we propose a Hebbian update rule using the above Hebbian theory to adapt the output network $f_\omega$ for fast-learning new classes. The weight $W \in \mathbb{R}^{d \times n}$ of $f_\omega$ can be seen as a set of $n$ vectors $w_i \in \mathbb{R}^d$, where $i \in \{1,...,n\}$ with each $i$ corresponds to a class. The Hebbian update rule for $w_i$ is defined as:

\[
\Delta_{w_i}^{Hebb} = \frac{1}{|N_i|} \sum_{k=1}^{|N_i|} c_k h_k,
\]  

(4)

where $N_i$ is the subset of entries in $N$ with class label $i$, $c_k$ is used for weighted update, and $1/|N_i|$ in Eq. (4) averages the cumulative effect of multiple entries with the same class label. A similar Hebbian update for the $i$-th element $(b_i)$ of the bias term of $f_\omega$ is:

\[
\Delta_{w_i}^{Hebb} = \frac{1}{|N_i|} \sum_{k=1}^{|N_i|} c_k.
\]

The Hebbian update rule in Eq. (4) applies the principle of Hebbian theory to the output layer by directly wiring the input representation $h_k$ and the corresponding label $y_k$ together, where $W, x_i^k, x_j^k$ in the Hebbian theory correspond to our $W, h_k, y_k$ respectively. The idea is to “memorize” representations of a new class in a sample-efficient manner by directly assigning them to the output network’s corresponding parameters. With the Hebbian update, the weights corresponding to a new class aligns with its representations to help the model predict the new class. The vectors in $W$ of which the corresponding classes are not in $N$ are not affected by the Hebbian update. Therefore, it only does a sparse update for parameters relevant to neighbors in $N$. A detailed analysis of the advantage of the proposed Hebbian update is included below.

3.3 Analysis and Comparison to Existing MbPA

The state-of-the-art parameter adaptation method $MbPA$ Sprechmann et al. [2018] is to adapt $f_\omega$ by maximizing the weighted log likelihood w.r.t. $N$ by:

\[
\max_{\omega} \mathcal{L}^N(f_\omega) = \max_{\omega} \frac{1}{|N|} \sum_{k=1}^{|N|} c_k \log P(y_k|h_k, \omega),
\]  

(5)

where $P(y_k) := P(y_k|h_k, \omega)$ is the predicted probability on true label $y_k$ for the $k$-th neighbor. The objective function is optimized by gradient descent, and one optimization step without considering learning rate can be written as: $\Delta \omega = -\nabla_{\omega} \mathcal{L}^N(f_\omega)$.

With the standard softmax activation with cross-entropy loss, the gradient contributed by the $k$-th neighbor with label $y_k$ w.r.t. to $w_i$ is $(P_{y_k} - \delta(i, y_k))h_k$, where $\delta()$ is the Kronecker delta. Therefore, the MbPA update with one optimization step for $w_i$ can be decomposed as:

\[
\Delta_{w_i}^{MbPA} = \Delta_{w_i}^N + \Delta_{w_i}^I
\]

\[
= -\nabla_{w_i} \mathcal{L}^N(f_\omega) - \nabla_{w_i} \mathcal{L}^I(f_\omega)
\]

\[
= \frac{1}{|N|} \sum_{k=1}^{|N_i|} c_k (1 - P_{y_k}) h_k - \frac{1}{|N|} \sum_{j=1}^{|N_i|} c_j P_{y_j} h_j,
\]

(6)

where we decompose $N$ into two disjoint sets $N_i$ and $N_i$, $N_i = \{(h_k, y_k = i, c_k)\}_{k=1}^{|N_i|}$ contains entries with label $i$, and $N_i = \{(h_j, y_j \neq i, c_j)\}_{j=1}^{|N_i|}$ contains entries with label different from $i$. $P_{y_k}$ is the predicted probability on the label $y_k$ of the $k$-th neighbor and similarly for $P_{y_j}$. $\Delta_{w_i}^N$ is the update from entries with label $i$, and $\Delta_{w_i}^I$ is the update from other entries.

For a new class $i$, effectively updating $W$ towards representations of $i$ is crucial for fast-learning this class. Next, we analyze why $\Delta_{w_i}^{Hebb}$ can better learn new classes than $\Delta_{w_i}^{MbPA}$ through the lens of the two terms in $\Delta_{w_i}^{MbPA}$.

- $\Delta_{w_i}^{Hebb}$ is more sensitive than the first term $\Delta_{w_i}^N$ of $\Delta_{w_i}^{MbPA}$ in terms of memorizing new class representations. For a new class $i$, $P_{y_k}$ is often very small

\(^1\)https://github.com/facebookresearch/faiss
where \( \omega \) is the parameter \((w_i \text{ or } b_i)\) of \( f_\omega \) for class \( i \in \{1, \ldots, n\} \), and \( \Delta \omega_i \) is the hybridized local adaptation for \( \omega_i \). \( n_i \) is the occurrence frequency of class \( i \) and \( \beta \in [0, 1) \) is a hyper-parameter controlling the decay rate of \( E_i \) as \( n_i \) increases. The idea is to rely more on the sparse Hebbian update (i.e., \( \Delta H^{Hebb}_\omega \)) when class \( i \) has not been seen many times. As it gradually becomes a frequent class, the adaptation relies more on the MbPA update.

The final prediction after parameter adaptation (during inference) is computed by \( \hat{y} = f_{\omega + \Delta \omega}(\mathbf{h}_x) \). The local adaptation \( \Delta \omega \) to \( f_\omega \) is discarded after the model makes a prediction, avoiding long-term overheads (e.g., overfitting).

### 3.5 Hebb Algorithm

The training and inference procedures of Hebb are given in Algorithm 1, and a detailed computation pipeline during inference is illustrated in Figure 1. Training data representations are stored during training to alleviate catastrophic forgetting, while the fast learning ability is achieved through parameter adaptation during inference.

- **Training procedure**: the feature extractor \( g_\theta \) and the output network \( f_\omega \) are trained first. Then, input representations and their corresponding labels of training data are stored to \( \mathcal{M} \).
- **Inference procedure**: the set of nearest neighbors \( \mathbf{N} \) of the input \( x \) is retrieved from \( \mathcal{M} \), and \( \Delta \omega^{MbPA} \) is computed w.r.t. \( \mathbf{N} \). When there are new classes to be learned after the initial training phase, we select a subset \( \mathcal{N}^{new} \) of \( \mathcal{N} \) whose labels are not seen during the initial training phase to compute \( \Delta H^{Hebb} \). Afterwards, these two updates are combined by Eq. (7) before a prediction \( \hat{y} = f_{\omega + \Delta \omega}(\mathbf{h}_x) \) is computed. In the cases of online adaptation during inference, e.g., evaluated in Section 4.3 and 4.4, \( \mathcal{M} \) is continuously updated with new testing data.

### 4 Experiments

As Hebb aims to learn new knowledge fast, the majority of our experiments study this aspect. We consider three learning scenarios for image classification. The continual learning setting in Section 4.1 briefly studies the catastrophic forgetting issue, while the incremental learning (Section 4.2) and online adaptation (Section 4.3) experiments study fast learning new classes. Section 4.4 studies an online adaptation setting for language model with different types of testing data (intra-domain and cross-domain).

For fairness, the memory component used by different methods is the same. The number of neighbors while us-

---

**Algorithm 1 Memory-based Hebbian Parameter Adaptation**

```plaintext
1: procedure TRAIN(training data: \( D_{train} \))
2:    Train \( g_\theta \) and \( f_\omega \) w.r.t \( D_{train} \)
3:    for \((x, y) \in D_{train}\) do
4:        Store \((g_\theta(x), y)\) into \( \mathcal{M} \)
5:    end for
6: end procedure
7: procedure INFERENCE(input: \( x \), ground truth: \( y \))
8:    Calculate input representation \( \mathbf{h}_x = g_\theta(x) \)
9:    Retrieve K-nearest neighbors \( \mathbf{N} \) of \( \mathbf{h}_x \) from \( \mathcal{M} \)
10:   Compute \( \Delta \omega^{MbPA} \) w.r.t. \( \mathbf{N} \)
11:   Select \( \mathcal{N}^{new} \) and compute \( \Delta H^{Hebb} \) w.r.t. \( \mathcal{N}^{new} \)
12:   Combine \( \Delta \omega^{MbPA} \) and \( \Delta H^{Hebb} \) by Eq. (7)
13:   Predict \( \hat{y} = f_{\omega + \Delta \omega}(\mathbf{h}_x) \)
14:   Store \((\mathbf{h}_x, y)\) into \( \mathcal{M} \) if "online adaptation"
15: end procedure
```
4.1 Continual Learning for Image Classification

In this experiment, we studied a continual learning setting to sequentially learn multiple tasks without forgetting previous ones. The “permuted MNIST” Goodfellow et al. [2013] dataset is used. Each task is given by a different random permutation (i.e., distribution shift) of the pixels of the MNIST dataset. We used a chain of 20 different tasks (20 different permutations) trained sequentially. Each task contains 10,000 samples. Different methods are trained 100 epochs for each task and evaluated on all tasks that have been trained on so far. The main challenge of this experiment is to prevent catastrophically forgetting image patterns in previous tasks.

For the base neural classifier, we use a one-layer MLP with size 1000. As in Sprechmann et al. [2018], we directly use pixels as input representation to query memory, i.e. an identity function for the feature extractor $g_\theta$, and the MLP serves as the output network $f_{\omega}$. The Hebbian update in Hebb is applied to all neighbors as no new classes are encountered. As baseline methods, we compared Hebb with the regular gradient descent training of MLP, EWC Kirkpatrick et al. [2017], and MbPA Sprechmann et al. [2018].

Results comparing different methods are included in Figure 2. As an effective approach to alleviate catastrophic forgetting, EWC performs much better than MLP. Both Hebb and MbPA perform better than EWC, which means that local parameter adaptation methods can recover classification performance when a task is catastrophically forgotten. The better performance of Hebb over MbPA demonstrates that Hebb effectively mitigate catastrophic forgetting.

### Table 1: Average Top-1 accuracy of the incremental image classification experiment.

| Model   | ResnetV1 Epoch 1 | ResnetV1 Epoch 3 | ResnetV1 Epoch 10 | Densenet Epoch 1 | Densenet Epoch 3 | Densenet Epoch 10 | MobilenetV2 Epoch 1 | MobilenetV2 Epoch 3 | MobilenetV2 Epoch 10 |
|---------|------------------|------------------|-------------------|------------------|------------------|-------------------|---------------------|---------------------|---------------------|
| **Overall** |                  |                  |                   |                  |                  |                   |                     |                     |                     |
| Parametric | 36.10%           | 41.38%           | 47.45%            | 34.48%           | 40.05%           | 45.14%            | 35.12%              | 40.64%              | 45.73%              |
| Mixure   | 38.62%           | 43.06%           | 47.74%            | 36.11%           | 41.74%           | 45.57%            | 35.72%              | 41.85%              | 45.95%              |
| MbPA     | 38.04%           | 45.58%           | 48.76%            | 36.25%           | 43.90%           | 47.53%            | 36.90%              | 43.51%              | 48.26%              |
| Hebb     | **39.04%**       | **47.16%**       | **49.69%**        | **37.07%**       | **45.80%**       | **48.02%**        | **37.72%**          | **45.85%**          | **48.69%**          |
| **New**  |                  |                  |                   |                  |                  |                   |                     |                     |                     |
| Parametric | 2.03%            | 19.15%           | 31.67%            | 1.63%            | 17.65%           | 29.94%            | 2.01%               | 18.05%              | 29.17%              |
| Mixure   | 7.08%            | 22.83%           | 32.05%            | 6.45%            | 21.06%           | 30.01%            | 6.95%               | 22.60%              | 30.85%              |
| MbPA     | 7.01%            | 27.96%           | 36.89%            | 6.55%            | 27.01%           | 35.92%            | 6.85%               | 22.60%              | 30.85%              |
| Hebb     | **10.26%**       | **31.92%**       | **39.01%**        | **9.75%**        | **30.23%**       | **38.10%**        | **10.02%**          | **31.45%**          | **38.70%**          |

Figure 2: Continual learning results to learn 20 tasks on the permuted MNIST dataset. 250/500 random samples are stored per task for MbPA and Hebb.

4.2 Incremental Learning for Image Classification

This experiment studied an incremental learning scenario to learn new knowledge. We considered the image classification task on the CIFAR100 Krizhevsky and others [2009] dataset. A neural classifier is pre-trained on 50 randomly selected image classes. Then during the incremental learning phase, the classifier is trained on an incremental training set containing all 100 classes (with 50 new classes not in the initial training set) and several baselines are compared during the incremental learning phase:

- **Parametric**: It fine-tunes the model on the incremental training set without using the memory component.
- **Mixture** Grave et al. [2017b,a]; Tu et al. [2018]; Orhan [2018]: It combines the prediction of Parametric with a non-parametric prediction from neighbors $\mathbb{N}$ by:
  \[
  P(y) \propto (1 - \gamma) e^{f_{\omega}(h_{t})} + \gamma \sum_{k=1}^{\lvert \mathbb{N} \rvert} I(y_k = y) e^{\theta h_{t}^{T} h_{t}}, \tag{8}
  \]
where $I(y_k = y)$ is the an indicator function, $\gamma$ controls the contribution of each part, and $\theta$ controls the flatness of the non-parametric prediction.
- **MbPA** Sprechmann et al. [2018]: It adapts the output network of Parametric using gradient descent to maximize Eq. (5) w.r.t. neighbors $\mathbb{N}$.
Hebb (proposed): Our scheme (c.f. Algorithm 1) dynamically combines the proposed Hebbian update with MbPA to adapt the output network of Parametric.

Class Balanced Incremental Learning Test accuracy of different methods on all 100 classes and 50 new classes are reported in Table 1. Performances on 50 old classes are not presented due to limited variations among different methods. Two interesting observations can be noted:

- **Hebb** achieves the best overall accuracy on all 100 classes at all epochs. Although MbPA notably outperforms both Parametric and Mixture, Hebb consistently outperforms MbPA at all epochs.
- **Hebb** learns new classes better. Hebb has evident improvements on 50 new classes, indicated by 3-5% gain over MbPA and 8-10% gain over Mixture and Parametric.
- **Hebb** learns new classes faster. The optimal accuracy on new classes achieved by Parametric and Mixture at epoch 10 can be obtained by Hebb within 3 epochs.

Class Imbalanced Incremental Learning CIFAR100 and most other datasets are artificially balanced; however, data imbalance is inevitable in most real-world applications. In this experiment, we follow the incremental learning experiment setup in Section 4.2 using ResnetV1, and we constructed two imbalanced incremental training sets. **Imbalanced level 2:1**: half of 50 new classes have twice samples as many as the other half. **Imbalanced level 5:1**: half of 50 new classes have five times samples as many as the other half. Models are still evaluated on the balanced test set of all 100 classes. The performances of different methods using ResnetV1 are presented in Figure 3 (Middle and Right). MbPA outperforms Parametric and Mixture with notable margins in the balanced setup (Figure 3-Left). However, the improvement margin is degraded in these two imbalanced setups. In contrast, Hebb is still consistently and notably better than MbPA at all epochs. This result reveals that Hebb is well suited to learn imbalanced new classes.

4.3 Online Adaptation for Image Classification

This online setting aims to evaluate the ability of a pre-trained model to learn new classes in an online manner. The pre-training phase is the same as the previous incremental learning experiment, in which the three base neural classifiers trained on 50 randomly sampled classes of CIFAR100. During online testing, we sequentially feed the complete test set with all 100 classes. The base classifier (Parametric) and the memory module used by Mixture, MbPA, and Hebb are updated as every 100 test samples arrive.

The average Top-1 accuracy after the online testing phase is summarized in Table 2. Different methods perform similarly on 50 old classes, with the simple Mixture method being slightly better. Hebb achieves the best overall performance on all 100 classes. It outperforms the closest (and SOTA) competitor MbPA by more than 1% and other baselines by larger margins. Furthermore, Hebb is especially good at learning 50 new classes. It outperforms MbPA by 2-3% and outperforms Mixture and Parametric by 3-5%.

Varying Number of Pre-training Classes In this experiment, we follow the online adaptation setup in Section 4.3, yet we vary the number of pre-training classes. Apart from 50 pre-training classes, we additionally report in Table 3 the overall performances of using 30 and 70 pre-training classes trained with ResnetV1. The fewer classes used for pre-training trained with ResnetV1. The fewer classes used for training highlighted improvements on 50 new classes, indicated by 3-5% gain over MbPA and 8-10% gain over Mixture and Parametric.
training, the more new classes need to be captured during the online testing phase. We can see from Table 3 that Hebb is consistently the best with a different number of pre-training classes. Furthermore, its improvement margin increases as more number of new classes need to be captured (e.g., when the number of pre-training classes decreases from 70 to 30). This result reinforces our conclusion that Hebb is especially good at learning new class patterns quickly.

For online image classification, we also include three extra experiments in Appendix B: (1) the hyper-parameter sensitivity of Hebb; (2) the computation efficiency of Hebb; (3) an ablation study analyzing the effect of different components of Hebb.

### 4.4 Online Adaptation for Language Model

Finally, we studied an online adaptation setting for the language model task to capture new vocabularies or distributions during test time. Two benchmark datasets are used, i.e., Penn Treebank (PTB) [Marcus et al. [1993]] and WikiText-2 [Merity et al. [2017]]. PTB is relatively small with vocabulary size 10,000. WikiText-2 from Wikipedia articles is larger with vocabulary size 33,278.

We consider two types of testing data. In the first intra-domain scenario, the testing data come from the same domain as the training data for pre-training with slight word distribution shifts. Because no new vocabularies are encountered, the Hebbian update of Hebb is computed over all entries in N. In the second cross-domain scenario, models are pre-trained on the training data of WikiText-2 and evaluated on the test set of PTB. This scenario contains both domain shifts and 3.77% out-of-vocabulary (OOV). We use a state-of-the-art LSTM (AWD-LSTM) [Merity et al. [2018]] as the base neural model. It is fixed during testing in the intra-domain scenario, and it is updated continually for every mini-batch (100 tokens in our case) in the more challenging cross-domain scenario. We reported perplexity (ppl.) and cross-entropy loss (CE-loss) for the two scenarios, respectively, because perplexities (exp(CE-loss)) on OOV in the cross-domain scenario are too large to be compared.

Table 4 summarizes the results of these two scenarios, and Table 5 further presents the test perplexity broken down by word frequency in the intra-domain scenario. Two observations can be noted. First, Hebb consistently achieves better overall performance than MbPA. The overall performance gain of Hebb over MbPA is mainly obtained from OOV in the cross-domain scenario and the less frequent vocabularies in the intra-domain scenario by inspecting Table 5. This observation validates that Hebb is especially effective to learn new vocabularies (OOV) or infrequent vocabularies. Second, Mix+Hebb achieves the best performance. Although the simple Mix method is very strong and it outperforms both MbPA and Hebb in most cases, hybridizing it with MbPA (Mix+MbPA) or with Hebb (Mix+Hebb) can consistently boost its performance. It means that MbPA and Hebb both have orthogonal benefits when combined with Mix.

### 5 Conclusion

This paper considers scenarios that require learning new classes or data distributions quickly without forgetting previous ones. To tackle the two major challenges (catastrophic forgetting, sample efficiency) towards this goal, we propose a method called “Memory-based Hebbian Parameter Adaptation” (Hebb). Hebb augments a regular neural classifier with a continuously updated memory module and a new parameter adaptation method based on the well-known Hebbian theory. Extensive experiments on a wide range of learning tasks (image classification, language model) and learning scenarios (continual, incremental, online) demonstrate the superior performance of Hebb.

### References

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In NeurIPS, pages 4331–4339, 2016.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In ECCV, pages 233–248, 2018.
A Reproducibility Checklist

A.1 Model Details and Hyper-parameters for Image Classification Experiments

Incremental Learning & Online Adaptation

During the initial model pre-training phase, different base neural classifiers (ResnetV1, MobilenetV2, and Densenet) are trained with SGD with momentum 0.9, learning rate 5e-4, and batch size 128. In total 350 epochs are trained, with a learning rate 0.1 in the first 150 epochs, 0.01 in the next 100 epochs, and 0.001 in the last 100 epochs.

After pre-training the neural classifiers, hyper-parameters of different baseline methods are tuned for the two learning scenarios (online and incremental) to maximize the overall performance on 100 classes. The hyper-parameter search space for different methods are:

- **Parametric**: We use RMSprop optimizer and tune the learning rate $Lr \in \{5e^{-5}, 1e^{-4}, 5e^{-4}, 1e^{-3}\}$.
- **Mixture**: Two weights $\theta \in \{0.4, 0.6, 0.8, 1\}$ and $\gamma \in \{0.05, 0.1, 0.2, 0.3\}$ are tuned and.
- **MbPA**: The learning rate $\lambda \in \{0.01, 0.02, 0.05, 0.1, 0.2\}$ and the number of optimization steps $\in \{1, 5, 10\}$ of the RMSprop optimizer are tuned without weight decay.
- **Hebb**: The learning rate $\eta \in \{0.5, 0.7, 0.9, 1.1, 1.3, 1.5\}$ and $\beta \in \{0.4, 0.5, 0.6, 0.7, 0.8\}$ of the dynamic weight term $E_y$ are tuned. It also re-uses the optimal hyper-parameters of MbPA.

The optimal hyper-parameters of different models and settings are presented in Table 6.

Continual Learning

We use the Adam Kingma and Ba [2015] as the optimizer with learning rate 1e-3 for MLP. The regularization term of EWC is set to be 1000. For MbPA, $\lambda$ is set to be 0.05, and the optimization step is set to 5. For Hebb, $\eta$ is set to 0.2 and $\beta$ is set to 0.9.

A.2 Model Details and Hyper-parameters for Language Model Experiments

For the base AWD-LSTM, we used 3 LSTM layers with a size 1200 each. For the initial model pre-training on PTB, the batch size is set to 20, the input layer dropout is set to 0.4, the hidden layer dropout is set to 0.25, and 500 epochs are trained. For the initial model pre-training on Wikitext-2, hidden layer dropout is set to 0.2, and 700 epochs are trained. Other configurations not mentioned are set by default according to the official implementation.

After the initial model pre-training phase, different methods are evaluated to learn distribution shifts or OOVs continually. We tune the hyper-parameters of different methods on corresponding validation sets to maximize overall perplexity on all vocabularies for both intra-domain and cross-domain scenarios. The hyper-parameter spaces to search for LSTM are:

1. https://github.com/salesforce/awd-lstm-lm

Automatic Table:

| Method            | Parameter 1 | Parameter 2 | Parameter 3 | Parameter 4 |
|-------------------|-------------|-------------|-------------|-------------|
| ResnetV1          | 5e-4        | 1           | 0.1         | 1e-3        |
| MobilenetV2       | 1e-3        | 1           | 0.05        | 1e-3        |
| Densenet          | 5e-4        | 1           | 0.05        | 1e-3        |
| Hebb              | 0.6         | 0.75        | 0.05        | 1e-3        |
| MbPA              | 0.5         | 0.5         | 0.05        | 1e-3        |

Table 6: Optimal hyper-parameters for different methods for image classification experiments in both incremental learning and online adaptation scenarios.

| Method            | Parameter 1 | Parameter 2 | Parameter 3 | Parameter 4 |
|-------------------|-------------|-------------|-------------|-------------|
| PTB               | -           | -           | 1e-4        |             |
| WikiText-2        | -           | -           | 1e-4        |             |
| LSTM (Lr)         | -           | -           |             |             |
| Mixture (\(\theta, \gamma\)) | (0.4, 0.01) | (0.4, 0.15) |             |             |
| MbPA (\(\lambda, \text{step}\)) | (0.1, 0.5) | (0.5, 1)   |             |             |
| Hebb (\(\beta, \eta\)) | (0.6, 0.7) | (0.15, 0.4) |             |             |
| Mixture + MbPA \(w\) | 0.05       | 0.15        |             |             |
| Mixture + Hebb \(w\) | 0.05       | 0.2         |             |             |

Table 7: Optimal hyper-parameters for different methods for language modeling experiments in both intra-domain and cross-domain settings.

Additional Experiment Results

B.1 Hyper-parameter Sensitivity of Hebb

We present a hyper-parameter sensitivity analysis of Hebb in the online image classification experiment. We can see from Figure 4 (Left) that the overall performance of Hebb is not sensitive to these two hyper-parameters: only four configurations of the red polygon are slightly worse than MbPA. These two hyper-parameters mainly affect the performance distribution to new and old classes, as illustrated in Figure 4 (Middle and Right). As $\eta$ increases and $\beta$ decreases, the performance on new classes increases, while the performance on old classes drops. Results reported in this paper were tuned to maximize the overall performance. However, readers can easily set these two hyper-parameters to favor either new or old classes.

B.2 Computation Efficiency of Hebb

In this experiment, we include the computation time of different methods of the online image classification experiment on CIFAR100 in Table 8. All methods are computed using a single GPU (GeForce GTX TITAN X). We can see that the computation overhead of Hebb on top of MbPA is marginal.
Figure 4: Hyperparameter sensitivity analysis of Hebb for the online image classification experiment on CIFAR100 using ResnetV1. (Left): overall Top 1 accuracy on 100 classes; (Middle): Top 1 accuracy on 50 new classes; (Right): Top 1 accuracy on 50 old classes. Cells within the red polygon are better than MbPA.

Table 8: The computation time (in seconds) of one online testing run for the online image classification experiment.

|          | ResnetV1 | Densenet | MobilenetV2 |
|----------|----------|----------|-------------|
| **Parametric** | 45.5     | 40.3     | 35.5        |
| **Mixture**  | 51.4     | 44.5     | 40.7        |
| **MbPA**    | 65.5     | 56.8     | 52.5        |
| **Hebb**    | 67.7     | 59.5     | 54.7        |

Table 9: Ablation study comparing Hebb with simplified versions in the online adaptation experiment using ResnetV1. Results on 50 new classes, 50 old classes, and all 100 classes are reported separately.

|          | New     | Old     | Overall  |
|----------|---------|---------|----------|
| MbPA     | 38.56%  | 73.56%  | 56.06%   |
| Hebb-v1  | 39.58%  | 71.94%  | 55.31%   |
| Hebb-v2  | 40.02%  | 72.64%  | 56.33%   |
| Hebb-v3  | 40.66%  | 72.84%  | 56.75%   |
| Hebb-250 | 39.91%  | 72.31%  | 56.11%   |
| Hebb-500 | 40.20%  | 72.45%  | 56.40%   |
| **Hebb** | **41.46%** | **73.16%** | **57.31%** |

B.3 Ablation study of Hebb

In this experiment, several simplified versions of Hebb are tested and compared to justify our design choices.

- **Hebb-v1**: It only uses the Hebbian update in Eq. (4) to adapt the output network’s parameters w.r.t. \( N^{\text{new}} \); the gradient-based MbPA update is discarded.

- **Hebb-v2**: It differs from Hebb by computing the Hebbian update w.r.t. \( N \), rather than \( N^{\text{new}} \).

- **Hebb-v3**: This version differs from Hebb by using a fixed static weight, rather than the dynamic weighting term \( E_{\omega} \), to merge \( \Delta_{\omega}^{\text{Hebb}} \) and \( \Delta_{\omega}^{\text{MbPA}} \).

- **Hebb-250/500**: This version implements a fixed-size memory by a ring buffer to store a limited number (250/500) of latest testing data.

Several observations can be noted from Table 9. **First**, the three variants (i.e. Hebb-v2, Hebb-v3, Hebb) that combines MbPA update (MbPA) with Hebbian update (Hebb-v1) all outperform the individual update variants. This result shows that combining these two update rules is better than using them separately. **Second**, the superior performance of Hebb over Hebb-v2 reveals the benefit of computing the Hebbian update w.r.t. \( N^{\text{new}} \). **Third**, the superior performance of Hebb over Hebb-v3 justifies the effectiveness of the proposed dynamic interpolation in eq. (7). **Fourth**, the superior performance of Hebb over Hebb-250/500 shows the benefit of using large memory sizes. Nevertheless, Hebb-200/500 (with limited memory size) is sufficient to outperform MbPA (unlimited memory size), especially on new classes.