Overcoming Catastrophic Forgetting with Hard Attention to the Task

Joan Serrà [1] **Dídac Surís** [1 2] **Marius Miron** [1 3] **Alexandros Karatzoglou** [1]

Abstract

Catastrophic forgetting occurs when a neural network loses the information learned in a previous task after training on subsequent tasks. This problem remains a hurdle for artificial intelligence systems with sequential learning capabilities. In this paper, we propose a task-based hard attention mechanism that preserves previous tasks' information without affecting the current task's learning. A hard attention mask is learned concurrently to every task, through stochastic gradient descent, and previous masks are exploited to condition such learning. We show that the proposed mechanism is effective for reducing catastrophic forgetting, cutting current rates by 45 to 80%. We also show that it is robust to different hyperparameter choices, and that it offers a number of monitoring capabilities. The approach features the possibility to control both the stability and compactness of the learned knowledge, which we believe makes it also attractive for online learning or network compression applications.

1. Introduction

With the renewed interest in neural networks, old problems re-emerge, specially if the solution is still open. That is the case with the so-called catastrophic forgetting or catastrophic interference problem (McCloskey & Cohen, 1989; Ratcliff, 1990). In essence, catastrophic forgetting corresponds to the tendency of a neural network to forget what it learned upon learning from new or different information. For instance, when a network is first trained to convergence on one task, and then trained on a second task, it forgets how to perform the first task.

Overcoming catastrophic forgetting is an important step in the advancement towards more general artificial intelligence systems (Legg & Hutter, 2007). Such systems should be able to seamlessly remember different tasks, and to learn them sequentially, following a lifelong learning paradigm (Thrun & Mitchell, 1995). Apart from being more biologically plausible (Clegg et al., 1998), there are many practical situations which require a sequential learning system (cf. Thrun & Mitchell, 1995). For instance, it may be unattainable for a robot to retrain from scratch its underlying model upon encountering a new object/task. After accumulating a large number of objects/tasks and their corresponding information, performing concurrent or multitask learning at scale may be too costly.

Storing previous information and using it to retrain the model was among the earliest attempts to overcome catastrophic forgetting; a strategy named "rehearsal" (Robins, 1995). The use of memory modules in this context has been a subject of research until today (Rebuffi et al., 2017; Lopez-Paz & Ranzato, 2017). However, due to efficiency and capacity constrains, memory-free approaches were also introduced, starting with what was termed as "pseudo-rehearsal" (Robins, 1995). This approach has found some success in transfer learning situations where one needs to maintain a certain accuracy on the source task after learning the target task (Jung et al., 2016; Li & Hoiem, 2017). Within the pseudo-rehearsal category, we could also consider recent approaches that substitute the memory module by a generative network (Venkatesan et al., 2017; Shin et al., 2017; Nguyen et al., 2017). Besides the difficulty of training a generative network for a sequence of tasks or certain types of data, both rehearsal and pseudo-rehearsal approaches imply some form of concurrent learning, that is, having to re-process 'old' instances for learning a new task.

The other popular strategy to overcome catastrophic forgetting is to reduce representational overlap (French, 1991). This can be done at the output, intermediate, and also input levels (Gutsein & Stump, 2015; He & Jaeger, 2018). A clean way of doing that in a soft manner is through so-called "structural regularization" (Zenke et al., 2017), either present in the loss function (Kirkpatrick et al., 2017; Zenke et al., 2017) or at a separate merging step (Lee et al., 2017). With these strategies, one seeks to prevent major changes in the weights that were important for previous tasks. Dedicating specific sub-parts of the network for each task is another

[1]Telefónica Research, Barcelona, Spain [2]Universitat Politècnica de Catalunya, Barcelona, Spain [3]Universitat Pompeu Fabra, Barcelona, Spain. Correspondence to: Joan Serrà <joan.serra@telefonica.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

way of reducing representational overlap (Rusu et al., 2016; Fernando et al., 2017; Yoon et al., 2018). The main trade-off in representational overlap is to effectively distribute the capacity of the network across tasks while maintaining important weights and reusing previous knowledge.

In this paper, we propose a task-based hard attention mechanism that maintains the information from previous tasks without affecting the learning of a new task. Concurrently to learning a task, we also learn almost-binary attention vectors through gated task embeddings, using backpropagation and minibatch stochastic gradient descent (SGD). The attention vectors of previous tasks are used to define a mask and constrain the updates of the network's weights on current tasks. Since masks are almost binary, a portion of the weights remains static while the rest adapt to the new task. We call our approach hard attention to the task (HAT). We evaluate HAT in the context of image classification, using what we believe is a high-standard evaluation protocol: we consider random sequences of 8 publicly-available data sets representing different tasks, and compare with a dozen of recent competitive approaches. We show favorable results in 4 different experimental setups, cutting current rates by 45 to 80%. We also show robustness with respect to hyperparameters and illustrate a number of monitoring capabilities. We make our code publicly-available[1].

2. Putting Hard Attention to the Task

2.1. Motivation

The primary observation that drives the proposed approach is that the task definition or, more pragmatically, its identifier, is crucial for the operation of the network. Consider the task of discriminating between bird and dog images. When training the network to do so, it may learn some set of intermediate features. If the second task is to discriminate between brown and black animals using the same data (assuming it only contained birds and dogs that were either brown or black), the network may learn a new set of features, some of them with not much overlap with the first ones. Thus, if training data is the same in both tasks, one important difference should be the task description or identifier. Our intention is to learn to use the task identifier to condition every layer, and to later exploit this learned conditioning to prevent forgetting previous tasks.

2.2. Architecture

To condition to the current task t, we employ a layer-wise attention mechanism (Fig. 1). Given the output of the units[2] of

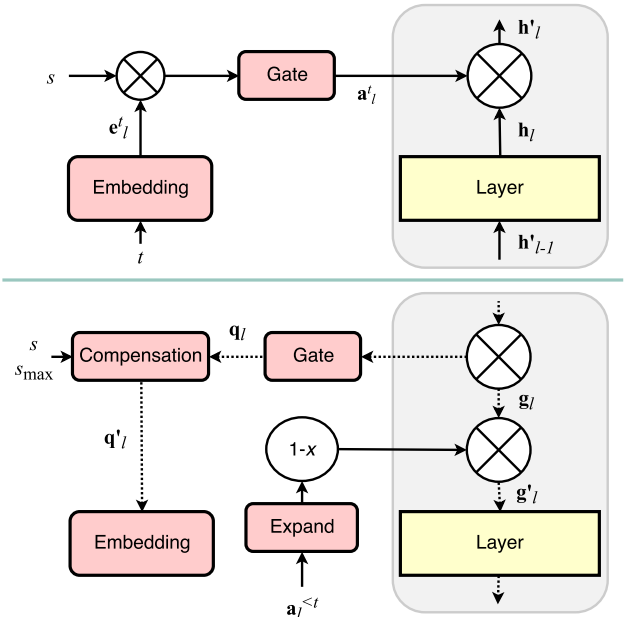

Figure 1. Schematic diagram of the proposed approach: forward (top) and backward (bottom) passes.

layer l, $\mathbf{h}_l$, we element-wise multiply $\mathbf{h}'_l = \mathbf{a}^t_l \odot \mathbf{h}_l$. However, an important difference with common attention mechanisms is that, instead of forming a probability distribution, $\mathbf{a}^t_l$ is a gated version of a single-layer task embedding $\mathbf{e}^t_l$,

$$\mathbf{a}^t_l = \sigma\left(s\mathbf{e}^t_l\right), \tag{1}$$

where $\sigma(x) \in [0, 1]$ is a gate function and s is a positive scaling parameter. We use a sigmoid gate in our experiments, but note that other gating mechanisms could be used. All layers $l = 1, \ldots L-1$ operate equally except the last one, layer L, where $\mathbf{a}^t_L$ is binary hard-coded. The operation of layer L is equivalent to a multi-head output (Bakker & Heskes, 2003), which is routinely employed in the context of catastrophic forgetting (for example Rusu et al., 2016; Li & Hoiem, 2017; Nguyen et al., 2017).

The idea behind the gating mechanism of Eq. 1 is to form hard, possibly binary attention masks which, act as "inhibitory synapses" (McCulloch & Pitts, 1943), and can thus activate or deactivate the output of the units of every layer. In this way, and similar to PathNet (Fernando et al., 2017), we dynamically create and destroy paths across layers that can be later preserved when learning a new task. However, unlike PathNet, the paths in HAT are not based on modules, but on single units. Therefore, we do not need to pre-assign a module size nor to set a maximum number of modules per task. Given some network architecture, HAT learns and automatically dimensions individual-unit paths, which ultimately affect individual layer weights. Furthermore, instead of learning paths in a separate stage using genetic algorithms, HAT learns them together with the rest of the network, using backpropagation and SGD.

[1] https://github.com/joansj/hat

[2] In the remaining of the paper, we will use 'units' to refer to both linear units (or fully-connected neurons) and convolutional filters. HAT can be extended to other parametric layers.

2.3. Network Training

To preserve the information learned in previous tasks upon learning a new task, we condition the gradients according to the cumulative attention from all the previous tasks. To obtain a cumulative attention vector, after learning task t and obtaining $\mathbf{a}_l^t$, we recursively compute

$$\mathbf{a}_l^{\leq t} = \max\left(\mathbf{a}_l^t, \mathbf{a}_l^{\leq t-1}\right),$$

using element-wise maximum and the all-zero vector for $\mathbf{a}_l^{\leq 0}$. This preserves the attention values for units that were important for previous tasks, allowing them to condition the training of future tasks.

To condition the training of task $t+1$, we modify the gradient $g_{l,ij}$ at layer l with the reverse of the minimum of the cumulative attention in the current and previous layers:

$$g'_{l,ij} = \left[1 - \min\left(a_{l,i}^{\leq t}, a_{l-1,j}^{\leq t}\right)\right] g_{l,ij}, \qquad (2)$$

where the unit indices i and j correspond to the output (l) and input ($l-1$) layers, respectively. In other words, we expand the vectors $\mathbf{a}_l^{\leq t}$ and $\mathbf{a}_{l-1}^{\leq t}$ to match the dimensions of the gradient tensor of layer l, and then perform a element-wise minimum, subtraction, and multiplication (Fig. 1). We do not compute any attention over the input data if this consists of complex signals like images or audio. However, in the case such data consisted of separate or independent features, one could also consider them as the output of some layer and apply the same methodology.

Note that, with Eq. 2, we create masks to prevent large updates to the weights that were important for previous tasks. This is similar to the approach of PackNet (Mallya & Lazebnik, 2017), which was made public during the development of HAT. In PackNet, after an heuristic selection and retraining, a binary mask is found and later applied to freeze the corresponding network weights. In this regard, HAT differs from PackNet in three important aspects. Firstly, our mask is unit-based, with weight-based masks automatically derived from those. Therefore, HAT also stores and maintains a lightweight structure. Secondly, our mask is learned, instead of heuristically- or rule-driven. Therefore, HAT does not need to pre-assign compression ratios nor to determine parameter importance through a post-training step. Thirdly, our mask is not necessarily binary, allowing intermediate values between 0 and 1. This can be useful if we want to reuse weights for learning other tasks, at the expense of some forgetting, or we want to work in a more online mode, forgetting the oldest tasks to remember new ones.

2.4. Hard Attention Training

To obtain a totally binary attention vector $\mathbf{a}_l^t$, one could use a unit step function as gate. However, since we want to train the embeddings $\mathbf{e}_l^t$ with backpropagation (Fig. 1), we prefer a differentiable function. To construct a pseudo-step function that allows the gradient to flow, we use a sigmoid with a positive scaling parameter s (Eq. 1). This scaling is introduced to control the polarization, or 'hardness', of the pseudo-step function and the resulting output $\mathbf{a}_l^t$. Our strategy is to anneal s during training, inducing a gradient flow, and set $s = s_{\max}$ during testing, using $s_{\max} \gg 1$ such that Eq. 1 approximates a unit step function. Notice that when $s \to \infty$ we get $a_{l,i}^t \to \{0, 1\}$, and that when $s \to 0$ we get $a_{l,i}^t \to 1/2$. We will use the latter to start a training epoch with all network units being equally active, and progressively polarize them within the epoch.

During a training epoch, we incrementally linearly anneal the value of s by

$$s = \frac{1}{s_{\max}} + \left(s_{\max} - \frac{1}{s_{\max}}\right)\frac{b-1}{B-1}, \qquad (3)$$

where $b = 1, \ldots B$ is the batch index and B is the total number of batches in an epoch. The hyperparameter $s_{\max} \geq 1$ controls the stability of the learned tasks or, in other words the plasticity of the network's units. If $s_{\max}$ is close to 1, the gating mechanism operates like a regular sigmoid function, without particularly enforcing the binarization of $\mathbf{a}_l^t$. This provides plasticity to the units, with the model being able to forget previous tasks at the backpropagation stage (Sec. 2.3). If, alternatively, $s_{\max}$ is a larger number, the gating mechanism starts operating as a unit step function. This provides stability with regard to previously learned tasks, preventing changes in the corresponding weights at the backpropagation stage.

2.5. Embedding Gradient Compensation

In preliminary analysis, we empirically observed that embeddings $\mathbf{e}_l^t$ were not changing much, and that the magnitude of the gradient was weak on those weights. After some investigation, we realized that the major part of the problem was due to the introduced annealing scheme (Eq. 3). To illustrate the effect of the annealing scheme on the gradients of $\mathbf{e}_l^t$, consider a uniformly distributed embedding $e_{l,i}^t$ across the active range of a standard sigmoid, $e_{l,i}^t \in [-6, 6]$. If we do not perform any annealing and set $s = 1$, we obtain a cumulative gradient after one epoch that has a bell-like shape and spans the whole sigmoid range (Fig. 2). Contrastingly, if we set $s = s_{\max}$, we obtain a much larger magnitude, but in a much lower range ($e_{l,i}^t \in [-1, 1]$ in Fig. 2). The annealed version of s yields a distribution in-between, with a lower range than $s = 1$ and a lower magnitude than $s = s_{\max}$. A desirable situation would be to have a wide range, ideally spanning the range of $s = 1$, and a large cumulative magnitude, ideally proportional to the one in the active region when $s = s_{\max}$. To achieve that, we apply a gradient compensation before updating $\mathbf{e}_l^t$.

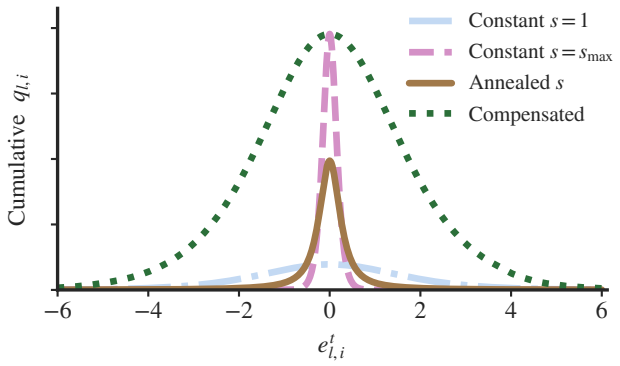

Figure 2. Illustration of the effect that annealing s has on the gradients q of $\mathbf{e}^t$.

In essence, the idea of the embedding gradient compensation is to remove the effects of the annealed sigmoid and to artificially impose the desired range and magnitude motivated in the previous paragraph. To do so, we divide the gradient $q_{l,i}$ by the derivative of the annealed sigmoid, and multiply by the desired compensation,

$$q'_{l,i} = \frac{s_{\max}\sigma\left(e^t_{l,i}\right)\left[1-\sigma\left(e^t_{l,i}\right)\right]}{s\sigma\left(se^t_{l,i}\right)\left[1-\sigma\left(se^t_{l,i}\right)\right]}\, q_{l,i},$$

which, after operating, yields

$$q'_{l,i} = \frac{s_{\max}\left[\cosh\left(se^t_{l,i}\right)+1\right]}{s\left[\cosh\left(e^t_{l,i}\right)+1\right]}\, q_{l,i}.$$

For numerical stability, we clamp $|se^t_{l,i}| \leq 50$ and constrain $e^t_{l,i}$ to remain within the active range of the standard sigmoid, $e^t_{l,i} \in [-6, 6]$. In any case, however, $q_{l,i} \to 0$ when we hit those limits. That is, we are in the constant regions of the pseudo-step function. Notice also that, by Eq. 3, the minimum s is never equal to 0.

2.6. Promoting Low Capacity Usage

It is important to realize that the hard attention values $a^t_{l,i}$ that are 'active', that is, $a^t_{l,i} \to 1$, directly determine the units that will be dedicated to task t. Therefore, in order to have some model capacity reserved for future tasks, we promote sparsity on the set of attention vectors $\mathsf{A}^t = \{\mathbf{a}^t_1, \ldots \mathbf{a}^t_{L-1}\}$. To do so, we add a regularization term to the loss function $\mathcal{L}$ that takes into account the set of cumulative attention vectors up to task $t-1$, $\mathsf{A}^{<t} = \{\mathbf{a}^{<t}_1, \ldots \mathbf{a}^{<t}_{L-1}\}$:

$$\mathcal{L}'\left(\mathbf{y}, \hat{\mathbf{y}}, \mathsf{A}^t, \mathsf{A}^{<t}\right) = \mathcal{L}\left(\mathbf{y}, \hat{\mathbf{y}}\right) + cR\left(\mathsf{A}^t, \mathsf{A}^{<t}\right), \quad (4)$$

where c is the regularization constant,

$$R\left(\mathsf{A}^t, \mathsf{A}^{<t}\right) = \frac{\sum_{l=1}^{L-1}\sum_{i=1}^{N_l} a^t_{l,i}\left(1-a^{<t}_{l,i}\right)}{\sum_{l=1}^{L-1}\sum_{i=1}^{N_l} 1-a^{<t}_{l,i}} \quad (5)$$

is the regularization term, and N_l corresponds to the number of units in layer l. Notice that Eq. 5 corresponds to a weighted and normalized L1 regularization over A^t. Cumulative attentions over the past tasks $\mathsf{A}^{<t}$ define a weight for the current task, such that if $a^{<t}_{l,i} \to 1$ then $a^t_{l,i}$ receives a weight close to 0 and vice versa. This excludes the units that were attended in previous tasks from regularization, unconstraining their reuse in the current task. The hyperparameter $c \geq 0$ controls the capacity spent on each task (Eq. 4). In a sense, it can be thought of as a compressibility constant, affecting the compactness of the learned models: the higher the c, the lower the number of active attention values $a^t_{l,i}$ and the more sparse the resulting network is. We set c globally for all tasks and let HAT adapt to the best compression for each individual task.

The use of L1 regularization to promote network sparsity in the context of catastrophic forgetting has also been considered by Yoon et al. (2018) with dynamically expandable networks (DEN), which were introduced while developing HAT. In DEN, plain L1 regularization is combined with a considerable set of heuristics such as L2-transfer, thresholding, and a measure of "semantic drift", and is applied to all network weights in the so-called "selective retraining" phase. In HAT, we use an attention-weighted L1 regularization over attention values, which is an independent part of the single training phase of the approach. Instead of considering network weights, HAT focuses on unit attention.

3. Related Work

We compare the proposed approach with the conceptually closest works, some of which appeared concurrently to the development of HAT. A more general overview of related work has been done in Sec. 1. A qualitative comparison with three of the most related strategies has been done along Sec. 2. A quantitative comparison with these and other approaches is done in Sec. 4 and Supplementary Materials.

Both elastic weight consolidation (EWC; Kirkpatrick et al., 2017) and synaptic intelligence (SI; Zenke et al., 2017) approaches add a 'soft' structural regularization term to the loss function in order to discourage changes to weights that are important for previous tasks. HAT uses a 'hard' structural regularization, and does it both at the loss function and gradient magnitudes explicitly. EWC measures weights' importance after network training, while SI and HAT compute weights' importance concurrently to network training. EWC and SI use specific formulation while HAT learns attention masks. Incremental moment matching (IMM; Lee et al., 2017) is an evolution of EWC, performing a separate model-merging step after learning a new task.

Progressive neural networks (PNNs; Rusu et al., 2016) distribute the network weights in a column-wise fashion, pre-

assigning a column width per task. They employ so-called adapters to reuse knowledge from previous columns/tasks, leading to a progressive increase of the number of weights assigned to future tasks. Instead of blindly pre-assigning column widths, HAT learns such 'widths' per layer, together with the network weights, and adapts them to the difficulty of the current task. PathNet (Fernando et al., 2017) also pre-assigns some amount of network capacity per task but, in contrast to PNNs, avoids network columns and adapters. It uses an evolutionary approach to learn paths between a constant number of so-called modules (layer subsets) that interconnect between themselves. HAT does not maintain a population of solutions, entirely trains with backpropagation and SGD, and does not rely on a constant set of modules.

Together with PNNs and PathNet, PackNet (Mallya & Lazebnik, 2017) also employs a binary mask to constrain the network. However, such constrain is not based on columns nor layer modules, but on network weights. Therefore, it allows for a potentially better use of the network's capacity. PackNet is based on heuristic weight pruning, with pre-assigned pruning ratios. HAT also focuses on network weights, but uses unit-based masks to constrain those, which also results in a lightweight structure. It avoids any absolute or pre-assigned pruning ratio, although it uses the compressibility parameter c to influence the compactness of the learned models. Another difference between HAT and the previous three approaches is that it does not use purely binary masks. Instead, the stability parameter $s_{\max}$ controls the degree of binarization.

Dynamically expandable networks (DEN; Yoon et al., 2018) also assign network capacity depending on the task at hand. However, they do so in a separate stage called "selective retraining". A complex mixture of heuristics and hyperparameters is used to identify "drifting" units, which are duplicated and retrained in another stage. L1 regularization and L2-transfer are employed to condition learning, together with the corresponding regularization constants and an additional set of thresholds. HAT strives for simplicity, restricting the number of hyperparameters to two that have a straightforward conceptual interpretation. Instead of plain L1 regularization over network weights, HAT employs an attention-weighted L1 regularization over attention masks. Attention masks are a lightweight structure that can be plugged in without the need of introducing important changes to a pre-existing network.

4. Experiments

Setups — Common setups to evaluate catastrophic forgetting in a classification context are based on permutations of the MNIST data (Srivastava et al., 2013), label splits of the MNIST data (Lee et al., 2017), incrementally learning classes of the CIFAR data sets (Lopez-Paz & Ranzato, 2017), or two-task transfer learning setups where accuracy is measured on both source and target tasks (Li & Hoiem, 2017). However, there are some limitations with these setups. Firstly, performing permutations of the MNIST data has been suggested to favor certain approaches, yielding misleading results[3] in the context of catastrophic forgetting (Lee et al., 2017). Secondly, using only the MNIST data may not be very representative of modern computer vision tasks, nor particularly challenging (Xiao et al., 2017). Thirdly, incrementally adding classes or groups of classes implies the assumption that all data comes from the same joint distribution, which is unrealistic for a real-world setting. Finally, evaluating catastrophic forgetting with only two tasks biases the conclusions towards transfer learning setups, and prevents the analysis of truly sequential learning with more than two tasks. In this paper, we consider the aforementioned MNIST and CIFAR setups (Sec. 4.2). Nonetheless, we primarily evaluate on a sequence of multiple tasks formed by different classification data sets (Sec. 4.1).

To obtain a generic estimate, we weigh a number of tasks and uniformly randomize their order. After training task t, we compute the accuracies on all testing sets of tasks $\tau \leq t$. We repeat 10 times this sequential train/test procedure with 10 different seed numbers, which are also used in the rest of randomizations and initializations (see below). To compare between different task accuracies, and in order to obtain a general measurement of the amount of forgetting, we introduce the forgetting ratio

$$\rho^{\tau\leq t} = \frac{A^{\tau\leq t} - A^{\tau}_{\text{R}}}{A^{\tau\leq t}_{\text{J}} - A^{\tau}_{\text{R}}} - 1, \tag{6}$$

where $A^{\tau\leq t}$ is the accuracy measured on task τ after sequentially learning task t, A^{τ}_{R} is the accuracy of a random stratified classifier using the class information of task τ, and $A^{\tau\leq t}_{\text{J}}$ is the accuracy measured on task τ after jointly learning t tasks in a multitask fashion. Note that $\rho \approx -1$ and $\rho \approx 0$ correspond to performances close to the ones of the random and multitask classifiers, respectively. To report a single number after learning t tasks, we take the average

$$\rho^{\leq t} = \frac{1}{t}\sum_{\tau=1}^{t} \rho^{\tau\leq t}.$$

Data — We consider 8 common image classification data sets and adapt them, if necessary, to an input size of $32 \times 32 \times 3$ pixels. The number of classes goes from 10 to 100, training set sizes from 16,853 to 73,257, and test set sizes from 1,873 to 26,032. For each task, we ran-

[3] Essentially, the MNIST data contains many values close to 0 that allow for an easier identification of the important units or weights which, if permuted, can then be easily frozen without overlapping with the ones of the other tasks (see Lee et al., 2017).

domly split 15% of the training set and keep it for validation purposes. The considered data sets are: CIFAR10 and CIFAR100 (Krizhevsky, 2009), FaceScrub (Ng & Winkler, 2014), FashionMNIST (Xiao et al., 2017), NotMNIST (Bulatov, 2011), MNIST (LeCun et al., 1998), SVHN (Netzer et al., 2011), and TrafficSigns (Stallkamp et al., 2011). For further details on data we refer to Supplementary Materials.

Baselines — We consider 2 reference approaches plus 9 recent and competitive ones: standard SGD with dropout (Goodfellow et al., 2014), SGD freezing all layers except the last one (SGD-F), EWC, IMM (Mean and Mode variants), learning without forgetting (LWF; Li & Hoiem, 2017), less-forgetting learning (LFL; Jung et al., 2016), PathNet, and PNNs. To find the best hyperparameter combination for each approach, we perform a grid search using a task sequence determined by a single seed. To compute the forgetting ratio ρ (Eq. 6), we also run the aforementioned random and multitask classifiers.

Network — Unless stated otherwise, we employ an AlexNet-like architecture (Krizhevsky et al., 2012) with 3 convolutional layers of 64, 128, and 256 filters with 4×4, 3×3, and 2×2 kernel sizes, respectively, plus two fully-connected layers of 2048 units each. We use rectified linear units as activations, and 2×2 max-pooling after the convolutional layers. We also use a dropout of 0.2 for the first two layers and of 0.5 for the rest. A fully-connected layer with a softmax output is used as a final layer, together with categorical cross entropy loss. All layers are randomly initialized with Xavier uniform initialization (Glorot & Bengio, 2010) except the embedding layers, for which we use a Gaussian distribution $\mathcal{N}(0, 1)$. Unless stated otherwise, our code uses PyTorch's defaults for version 0.2.0 (Paszke et al., 2017). We adapt the same base architecture to all baseline approaches and match their number of parameters to 7.1 M.

Training — We train all models with backpropagation and plain SGD, using a learning rate of 0.05, and decaying it by a factor of 3 if there is no improvement in the validation loss for 5 consecutive epochs. We stop training when we reach a learning rate lower than 10^{-4} or we have iterated over 200 epochs (we made sure that all considered approaches reached a stable solution before 200 epochs). Batch size is set to 64. All methods use the same task sequence, data split, batch shuffle, and weight initialization for a given seed.

4.1. Results

We first look at the average forgetting ratio $\rho^{\leq t}$ after learning task t (Fig. 3). A first thing to note is that not all the considered baselines perform better than the SGD references. That is the case of LWF and LFL. For LWF, we observe it is still competitive in the two-task setup for which it was designed, $t = 2$. However, its performance rapidly degrades for $t > 2$, indicating that the approach has difficulties in extending

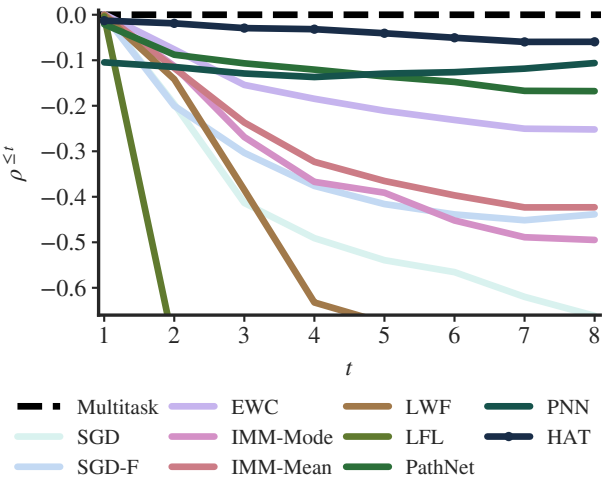

Figure 3. Average forgetting ratio $\rho^{\leq t}$ for the considered approaches (10 runs).

Table 1. Average forgetting ratio after the second ($\rho^{\leq 2}$) and the last ($\rho^{\leq 8}$) task for the considered approaches (10 runs, standard deviation into parenthesis).

APPROACH	$\rho^{\leq 2}$	$\rho^{\leq 8}$
LFL	-0.73 (0.29)	-0.92 (0.08)
LWF	-0.14 (0.13)	-0.80 (0.06)
SGD	-0.20 (0.08)	-0.66 (0.03)
IMM-MODE	-0.11 (0.08)	-0.49 (0.05)
SGD-F	-0.20 (0.15)	-0.44 (0.06)
IMM-MEAN	-0.12 (0.10)	-0.42 (0.04)
EWC	-0.08 (0.06)	-0.25 (0.03)
PATHNET	-0.09 (0.16)	-0.17 (0.23)
PNN	-0.11 (0.10)	-0.11 (0.01)
HAT	**-0.02 (0.03)**	**-0.06 (0.01)**

beyond a transfer learning setup. We find LFL extremely sensitive to the configuration of its hyperparameter, to the point that what is a good value for one seed, turns out to be a bad choice for another seed. Hence the poor average performance for 10 seeds. The highest standard deviations are obtained by LFL and PathNet (Table 1), which suggests a high sensitivity with respect to hyperparameters, initializations, or data sets. Another thing to note is that the IMM approaches only perform similarly or slightly better than the SGD-F reference. We believe this is due to both the different nature of the tasks' data and the consideration of more than two tasks, which complicates the choice of the mixing hyperparameter.

The best performing baselines are EWC, PathNet, and PNN. PathNet and PNN present contrasting behaviors. Both, by construction, never forget; therefore, the important difference is in their learning capability. PathNet starts by correctly learning the first task and progressively exhibits difficulties to do so for $t \geq 2$. Contrastingly, PNNs exhibits difficulty in the first tasks and becomes better as t increases.

These contrasting behaviors are due to the way the two approaches allocate the network capacity. As mentioned, they cannot do it dynamically, and therefore need to pre-assign a number of network weights per task. When having more tasks but the same network capacity, this pre-assignment increasingly harms the performance of these baselines, lowering the corresponding curves in Fig. 3.

We now move to the HAT results. First of all, we observe that HAT consistently performs better than all considered baselines for all $t \geq 2$ (Fig. 3). For the case of $t = 2$, it obtains an average forgetting ratio $\rho^{\leq 2} = -0.02$, while the best baseline is EWC with $\rho^{\leq 2} = -0.08$ (Table 1). For the case of $t = 8$, HAT obtains $\rho^{\leq 8} = -0.06$, while the best baseline is PNN with $\rho^{\leq 8} = -0.11$. This implies a reduction in forgetting of 75% for $t = 2$ and 45% for $t = 8$. Notice that the standard deviation of HAT is lower than the ones obtained by the big majority of the baselines (Table 1). This denotes a certain stability of HAT with respect to different task sequences, data sets, data splits, and network initializations.

Given the slightly increasing tendency of PNN with t (Fig. 3), one could speculate that PNN would score above HAT for $t > 8$. However, our empirical analyzes suggest that that is not the case (presumably due to the capacity pre-assignment and parameter increase problems underlined in Sec. 3 and above). In particular, we observe a gradual lowering of PathNet and PNN curves with increasing sequences from $t = 2$ to 8. In addition, we observe PathNet and PNN obtaining worse performances than EWC in the case of $t = 10$ for the incremental class setup (see below and Supplementary Materials). In general, none of the baseline methods consistently outperforms the rest across setups and for all t, a situation that we do observe with HAT.

4.2. Additional Results

To broaden the strength of our results, we additionally experiment with three common alternative setups. First, we consider an incremental class learning scenario, similar to Lopez-Paz & Ranzato (2017), using class subsets of both CIFAR10 and CIFAR100 data. In this setup, the best baseline after $t \geq 3$ is EWC, with $\rho^{\leq 10} = -0.18$. HAT scores $\rho^{\leq 10} = -0.09$ (55% forgetting reduction). Next, we consider the permuted MNIST sequence of tasks (Srivastava et al., 2013). In this setup, the best result we could find in the literature was from SI, with $A^{\leq 10} = 97.1\%$. HAT scores $A^{\leq 10} = 98.6\%$ (52% error rate reduction). Finally, we also consider the split MNIST task of Lee et al. (2017). In this setup, the best result from the literature corresponds to the conceptor-aided backpropagation approach (He & Jaeger, 2018), with $A^{\leq 2} = 94.9\%$. HAT scores $A^{\leq 2} = 99.0\%$ (80% error rate reduction). The detail for all these setups and results can be found in Supplementary Materials.

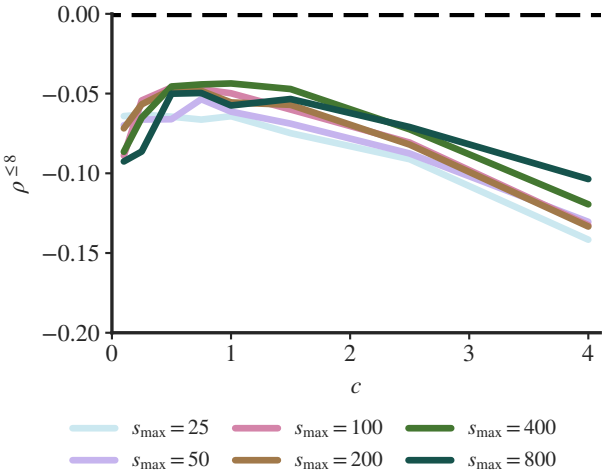

Figure 4. Effect of hyperparameters $s_{\max}$ and c on average forgetting ratio $\rho^{\leq 8}$. Results for seed 0.

4.3. Hyperparameters

In any machine learning algorithm, it is important to assess the sensitivity with respect to the hyperparameters. HAT has two: the stability parameter $s_{\max}$ and the compressibility parameter c (Secs. 2.4 and 2.6). A low $s_{\max}$ provides plasticity to the units and capacity of adaptation, but the network may easily forget what it learned. A high $s_{\max}$ prevents forgetting, but the network may have difficulties in adapting to new tasks. A low c allows to use almost all of the network's capacity for a given task, potentially spending too much in the current task. A high c forces it to learn a very compact model, at the expense of not reaching the accuracy that the original network could have reached. We empirically found good operation ranges $s_{\max} \in [25, 800]$ and $c \in [0.1, 2.5]$. As we can see, any variation within these ranges results in reasonable performance (Fig. 4). Unless stated otherwise, we use $s_{\max} = 400$ and $c = 0.75$.

4.4. Monitoring and Network Pruning

It is interesting to note that the hard attention mechanism introduced in Sec. 2 offers a number of possibilities to monitor the behavior of our models. For instance, by computing the conditioning mask in Eq. 2 from the hard attention vectors $\mathbf{a}_l^{\leq t}$, we can assess which weights obtain a high attention value, binarize it, and compute an estimate of the instantaneous network capacity usage (Fig. 5). We may also inform ourselves of the amount of active weights per layer and task (Supplementary Materials). Another facet we can monitor is the weight reuse across tasks. By a similar procedure, comparing the conditioning masks between tasks t_i and t_j, $j > i$, we can asses the percentage of weights of task t_i that are later reused in task t_j (Fig. 6).

Another by-product of hard attention masks is that we can

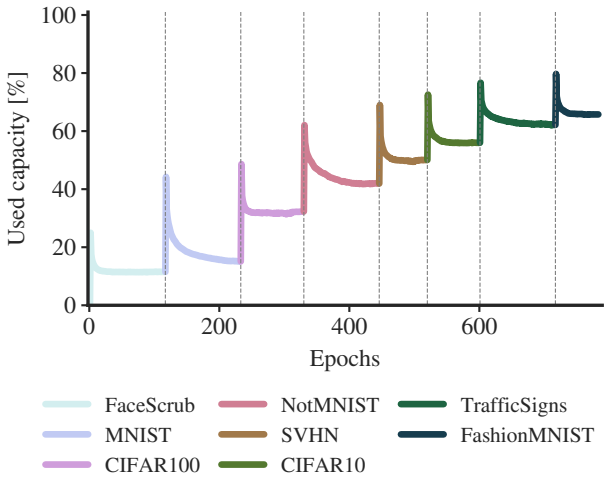

Figure 5. Network capacity usage with sequential task learning (seed 0). Dashed vertical lines correspond to a task switch.

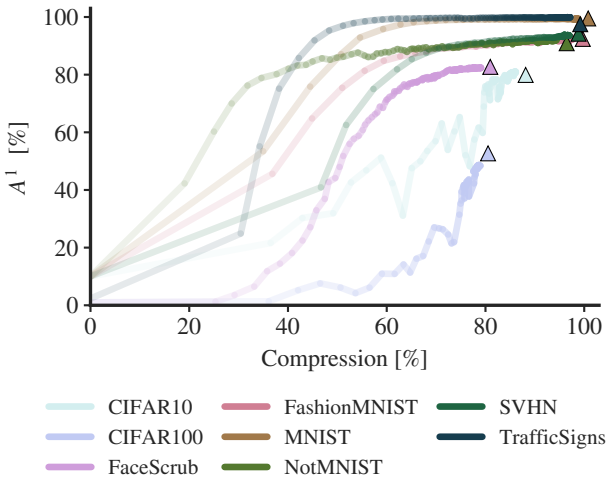

Figure 7. Validation accuracy A^1 as a function of compression percentage. Every dot corresponds to an epoch and triangles match the accuracy of the SGD approach (no compression).

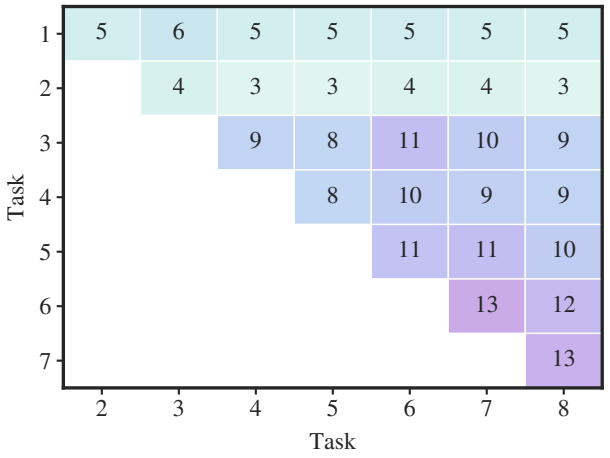

Figure 6. Percentage of weight reuse across tasks. Seed 0 sequence: (1) FaceScrub, (2) MNIST, (3) CIFAR100, (4) NotMNIST, (5) SVHN, (6) CIFAR10, (7) TrafficSigns, and (8) FashionMNIST.

use them to assess which of the network's weights are important, and then prune the most irrelevant ones (LeCun et al., 1990). This way, we can compress the network for further deployment in low-resource devices or time-constrained environments (cf. Han et al., 2016). If we want to focus on such compression task, we can set c to a higher value than the one used for catastrophic forgetting and start with a positive random initialization of the embeddings $\mathbf{e}_l$. The former will promote more compression while the latter will ensure we start learning the model by putting attention to all weights in the first epochs (full capacity). We empirically found that using $c = 1.5$ and $\mathcal{U}(0, 2)$ yields a reasonable trade-off between accuracy and compression for a single task (Fig. 7). With that, we can compress the network to sizes between 1 and 21% of its original size, depending on the task (Supplementary Materials). Comparing these numbers with the compression rates used by PackNet (25 or 50%), we see that HAT generally uses a much more compact model. Comparing with DEN on the specific MNIST and CIFAR100 tasks (18 and 52%), we observe that HAT compresses to 1 and 21%, respectively. Interestingly, and in contrast to these and the majority of network pruning approaches, HAT learns to prune network weights through backpropagation and SGD, and at the same time as the network weights themselves.

5. Conclusion

We introduce HAT, a hard attention mechanism that, by focusing on a task embedding, is able to protect the information of previous tasks while learning new tasks. This hard attention mechanism is lightweight, in the sense that it adds a small fraction of weights to the base network, and is trained together with the main model, with negligible overhead using backpropagation and vanilla SGD. We demonstrate the effectiveness of the approach to control catastrophic forgetting in the image classification context by running a series of experiments with multiple data sets and state-of-the-art approaches. HAT has only two hyperparameters, which intuitively refer to the stability and compactness of the learned knowledge, and whose tuning we demonstrate is not crucial for obtaining good performance. In addition, HAT offers the possibility to monitor the used network capacity across tasks and layers, the unit reuse across tasks, and the compressibility of a model trained for a given task. We hope that our approach may be also useful in online learning or network compression contexts, and that the hard attention mechanism presented here may also find some applicability beyond the catastrophic forgetting problem.

References

Bakker, B. and Heskes, T. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 4:83–99, 2003.

Bulatov, Y. NotMNIST dataset. Technical report, 2011. URL http://yaroslavvb.blogspot.it/2011/09/notmnist-dataset.html.

Clegg, B. A., DiGirolamo, G. J., and Keele, S. W. Sequence learning. *Trends in Cognitive Sciences*, 2(8):275–281, 1998.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. PathNet: evolution channels gradient descent in super neural networks. *ArXiv*, 1701.08734, 2017.

French, R. M. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. In *Proc. of the Annual Conf. of the Cognitive Science Society (CogSci)*, pp. 173–178, 1991.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proc. of the Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, pp. 249–256, 2010.

Goodfellow, I., Mizra, M., Da, X., Courville, A., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2014.

Gutsein, S. and Stump, E. Reduction of catastrophic forgetting with transfer learning and ternary output codes. In *Proc. of the Int. Joint Conf. on Neural Networks (IJCNN)*, pp. 1–8, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2016.

He, X. and Jaeger, H. Overcoming catastrophic interference using conceptor-aided backpropagation. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2018.

Jung, H., Ju, J., Jung, M., and Kim, J. Less-forgetting learning in deep neural networks. *ArXiv*, 1607.00122, 2016.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *Proc. of the National Academy of Sciences of the USA*, 114(13):3521–3526, 2017.

Krizhevsky, A. *Learning multiple layers of features from tiny images*. Msc thesis, University of Toronto, Toronto, Canada, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. ImageNet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 25, pp. 1097–1105. 2012.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems (NIPS)*, volume 2, pp. 598–605. Morgan Kaufmann, 1990.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., and Zhang, B.-T. Overcoming catastrophic forgetting by incremental moment matching. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pp. 4655–4665. Curran Associates Inc., 2017.

Legg, S. and Hutter, M. Universal intelligence: a definition of machine intelligence. *Minds and Machines*, 17(4): 391–444, 2007.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PP (99):1–1, 2017.

Lopez-Paz, D. and Ranzato, M. A. Gradient episodic memory for continuum learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pp. 6449–6458. Curran Associates Inc., 2017.

Mallya, A. and Lazebnik, S. PackNet: adding multiple tasks to a single network by iterative pruning. *ArXiv*, 1711.05769, 2017.

McCloskey, M. and Cohen, N. Catastrophic interference in connectionist networks: the sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.

McCulloch, W. S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning (NIPS-DeepLearning)*, 2011.

Ng, H.-W. and Winkler, S. A data-driven approach to cleaning large face datasets. In *Proc. of the IEEE Int. Conf. on Image Processing (ICIP)*, pp. 343–347, 2014.

Nguyen, C., Li, Y., Bui, T. D., and Turner, R. E. Variational continual learning. *ArXiv*, 1710.10628, 2017.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. In *NIPS Workshop on The Future of Gradient-based Machine Learning Software & Techniques (NIPS-Autodiff)*, 2017.

Ratcliff, R. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological Review*, 97:285–308, 1990.

Rebuffi, S., Kolesnikov, A., Sperl, G., and Lampert, C. iCaRL: incremental classifier and representation learning. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 2001–2010, 2017.

Robins, A. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7:123–146, 1995.

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

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pp. 2993–3002. Curran Associates Inc., 2017.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., and Schmidhuber, J. Compete to compute. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pp. 2310–2318. Curran Associates Inc., 2013.

Stallkamp, J., Schlipsing, M., Salmen, J., and Igel, C. The German traffic sign recognition benchmark: a multi-class classification competition. In *Proc. of the Int. Joint Conf. on Neural Networks (IJCNN)*, pp. 1453–1460, 2011.

Thrun, S. and Mitchell, T. Lifelong robot learning. *Robotics and Autonomous Systems*, 15:25–46, 1995.

Venkatesan, R., Venkateswara, H., Panchanathan, S., and Li, B. A strategy for an uncompromising incremental learner. *ArXiv*, 1705.00744, 2017.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, 1708.07747, 2017.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong learning with dynamically expandable networks. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2018.

Zenke, F., Poole, B., and Ganguli, S. Improved multitask learning through synaptic intelligence. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pp. 3987–3995, 2017.