OVERCOMING NEURAL BRAINWASHING

Anonymous authors
Paper under double-blind review

ABSTRACT

We identify a phenomenon, which we dub *neural brainwashing*, that occurs when sequentially training multiple deep networks with partially-shared parameters; the performance of previously-trained models degrades as one optimizes a subsequent one, due to the overwriting of shared parameters. To overcome this, we introduce a statistically-justified weight plasticity loss that regularizes the learning of a model's shared parameters according to their importance for the previous models, and demonstrate its effectiveness when training two models sequentially and for neural architecture search. Adding weight plasticity in neural architecture search preserves the best models to the end of the search process leading to improved results in both natural language processing and computer vision tasks.

1 INTRODUCTION

Deep neural networks have been very successful for tasks such as visual recognition (Xie & Yuille, 2017) and natural language processing (Young et al., 2017), and much recent work has addressed the training of models that can generalize across multiple tasks (Caruana, 1997). In this context, when the tasks become available sequentially, a major challenge is *catastrophic forgetting*: when a model initially trained on task A is later trained on task B, its performance on task A can decline calamitously. Many recent articles have addressed this problem (Kirkpatrick et al., 2017; Rusu et al., 2016; He & Jaeger, 2017; Li & Hoiem, 2016). In particular, Kirkpatrick et al. (2017) show how to overcome catastrophic forgetting by approximating the posterior probability, $p(\boldsymbol{\theta} \mid \mathcal{D}_1, \mathcal{D}_2)$, with $\boldsymbol{\theta}$ the network parameters and $\mathcal{D}_1, \mathcal{D}_2$ different datasets representing the tasks.

In many situations one does not train *a single model for multiple tasks* but *multiple models for a single task*. When dealing with many large models, a common strategy for training to remain tractable involves sharing a subset of the weights across the multiple models and training them sequentially (Pham et al., 2018; Xie & Yuille, 2017; Liu et al., 2018a). This, however, has an important drawback. Figure 1 shows that for two models, A and B, the larger the number of shared weights, the more the accuracy of A drops when training B; B overwrites some of the weights of A and this badly damages the performance of A. We dub this *neural brainwashing*.

While *forgetting* is an individual problem, where one model struggles to maintain internal cohesion while absorbing information about novel tasks, *brainwashing* is inherently "social": in neural terms, one model moves another's weights away from their optimum. The benefits of weight sharing have been emphasized in tasks like neural architecture search, where the associated speed gains have been key in making the process practical (Pham et al., 2018; Liu et al., 2018b), but its downsides have not been investigated.

In this paper we introduce an approach to overcoming neural brainwashing. Given a dataset $\mathcal{D}$, we first consider two models $f_1(\mathcal{D}; \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$ and $f_2(\mathcal{D}; \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)$ with shared weights $\boldsymbol{\theta}_s$ and private weights $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. We formulate learning as the maximization of the posterior $p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_s | \mathcal{D})$. Under mild assumptions we show that this posterior can be approximated and expressed using a loss, dubbed Weight Plasticity Loss (WPL), that minimizes neural brainwashing. Our framework evaluates the importance of each weight, conditioned on the previously-trained model, and encourages the update of each shared weight to be inversely proportional to its importance. We then show that our approach extends to more than two models by exploiting it for neural architecture search.

Our work is the first of which we are aware to identify neural brainwashing and to propose a solution. We establish the merits of our approach when training two models with partially shared weights and in the context of neural architecture search. For the former, we establish the effectiveness of WPL in

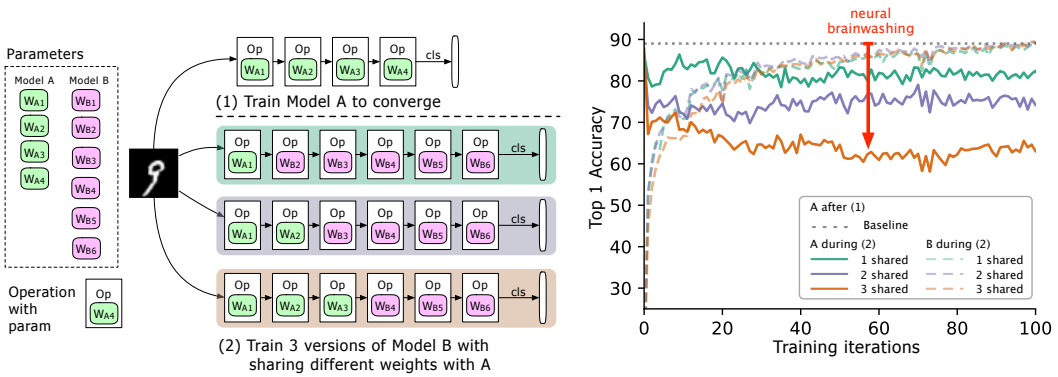

Figure 1: *(Left)* Two models to be trained (A, B), where A's parameters are in green and B's in purple, and B shares some parameters with A (indicated in green during phase 2). We first train A to convergence and then train B. *(Right)* Accuracy of model A as the training of B progresses. The different colors correspond to different numbers of shared layers. The accuracy of A decreases dramatically, especially when more layers are shared, and we dub the drop (the red arrow) *neural brainwashing*. This experiment was performed on MNIST (LeCun & Cortes, 2010).

the strict convergence case, where each model is trained until convergence, and in the more realistic loose convergence setting, where training is stopped early. WPL can reduce the brainwashing effect by 99% when model A converges fully, and by 52% in the loose convergence case.

For neural architecture search, we implement WPL within the efficient ENAS method of Pham et al. (2018), a state-of-the-art architecture search technique that relies on parameter sharing and corresponds to the loose convergence setting. We show that, at each iteration, the use of WPL reduces the brainwashing effect by 51% on the most affected model and by 95% on average over all sampled models. Our final results on the best architecture found by the search confirm that limiting brainwashing is needed to achieve better results and better convergence for both language modeling (on the PTB dataset (Marcus et al., 1994)) and image classification (on the CIFAR10 dataset (Krizhevsky et al., 2009)). For language modeling the perplexity decreases from 65.01 for ENAS without WPL to 61.9 with WPL. For image classification WPL yields a drop of top-1 error from 4.87% to 3.81%. We will make our code publicly available upon acceptance of this paper.

2 Related Work

Single-model Forgetting. The goal of training a single model to tackle multiple problems is to leverage the structures learned for one task for other tasks. This has been employed in transfer learning (Pan & Yang, 2010), multi-task learning (Caruana, 1997) and lifelong learning (Silver et al., 2013). However, sequential learning of later tasks has visible negative consequences for the initial one. Kirkpatrick et al. (2017) selectively slow down the learning of the weights that are comparatively important for the first task by defining the importance of an individual weight using the Fisher information for this parameter (Rissanen, 1996). He & Jaeger (2017) projects the gradient so that directions relevant to the previous task are unaffected. Other families of methods save the older models separately to create progressive networks (Rusu et al., 2016) or regularize the parameters to remain close to the values obtained by previous tasks while learning new ones (Li & Hoiem, 2016). An extreme case of sequential learning is lifelong learning, for which the solution to catastrophic forgetting developed by Aljundi et al. (2018) is also to prioritize the weight updates, with smaller updates for weights that are important for previously-learned tasks.

Parameter Sharing in Neural Architecture Search. In both sequential learning on multiple tasks and lifelong learning, forgetfulness is the problem of an individual model. Here we tackle scenarios where one seeks to optimize a population of multiple models that share parts of their internal structure. The use of multiple models to solve a single task dates back to model ensembles (Dietterich, 2000). Recently, sharing weights between models that are candidate solutions to a problem has shown great promise in the generation of custom neural architectures, known as neural architecture search (Elsken et al., 2018). Existing neural architecture search strategies mostly divide into reinforcement learning and evolutionary techniques. For instance, Zoph & Le (2017) use reinforcement learning to explore a search space of candidate architectures, with each architecture encoded as a

string using an RNN trained with REINFORCE (Williams, 1992) and taking validation performance as the reward. MetaQNN (Baker et al., 2017) uses Q-Learning to design CNN architectures. By contrast, neuro-evolution strategies use evolutionary algorithms (Bäck, 1996) to perform the search. An example is Liu et al. (2018a), who introduce a hierarchical representation of neural networks and use tournament selection (Goldberg & Deb, 1991) to evolve the architectures.

Initial search solutions required hundreds of GPUs due to the huge search space, but recent efforts have made the search more tractable, for example via the use of neural blocks (Negrinho & Gordon, 2017; Bennani-Smires et al., 2018). Similarly, and directly related to this work, *weight sharing* between the candidates has allowed researchers to greatly decrease the computational cost of neural architecture search. For neuroevolution methods, sharing is implicit. For example, Real et al. (2017) define weight inheritance as allowing the children to inherit their parents' weights whenever possible. For RL-base techniques, weight sharing is modeled explicitly and has been shown to lead to significant gains. In particular, ENAS (Pham et al., 2018), which builds upon NAS (Zoph & Le, 2017), represents the search space as a single directed acyclic graph (DAG) in which each candidate architecture is a subgraph. EAS (Cai et al., 2018) also uses an RL strategy to grow the network depth or layer width with function-preserving transformations defined by Chen et al. (2016) where they initialize new model with previous parameters.

DARTS (Liu et al., 2018b) uses soft assignment to select paths that implicitly inherit the previous weights. While weight sharing has proven effective, its downsides have not been studied. This is what we contribute with this paper, together with a solution to these downsides.

3 METHODOLOGY

In this section we study the training of multiple models that share certain parameters. As discussed above, training the multiple models sequentially as in Pham et al. (2018), for example, is suboptimal, since neural brainwashing arises. Below we derive a method to overcome neural brainwashing for two models, and then show how our formalism extends to multiple models in the context of neural architecture search, and in particular within the state-of-the-art ENAS of Pham et al. (2018).

3.1 WEIGHT PLASTICITY LOSS: PREVENTING NEURAL BRAINWASHING

Given a dataset $\mathcal{D}$, we seek to train two architectures $f_1(\mathcal{D}; \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$ and $f_2(\mathcal{D}; \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)$ with shared parameters $\boldsymbol{\theta}_s$ and private parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. We suppose that the models are trained sequentially, which reflects common large-model, large-dataset scenarios and will facilitate generalization. Below, we derive a statistically-motivated framework that prevents neural brainwashing; it stops the training of the second model from degrading the performance of the first model.

We formulate training as finding the parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)$ that maximize the posterior probability $p(\boldsymbol{\theta} \mid \mathcal{D})$, which we approximate to derive our new loss function. Below we discuss the different steps of this approximation, first expressing $p(\boldsymbol{\theta} \mid \mathcal{D})$ more conveniently.

Lemma 1. *Given a dataset $\mathcal{D}$ and two architectures with shared parameters $\boldsymbol{\theta}_s$ and private parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, and provided that $p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \mid \boldsymbol{\theta}_s, \mathcal{D}) = p(\boldsymbol{\theta}_1 \mid \boldsymbol{\theta}_s, \mathcal{D}) p(\boldsymbol{\theta}_2 \mid \boldsymbol{\theta}_s, \mathcal{D})$, we have*

$$p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_s \mid \mathcal{D}) \propto \frac{p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s) p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s)}{\int p(\mathcal{D} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_s) p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) d\boldsymbol{\theta}_1}. \tag{1}$$

Proof. Provided in the appendix. □

Lemma 1 presupposes that $p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \mid \boldsymbol{\theta}_s, \mathcal{D}) = p(\boldsymbol{\theta}_1 \mid \boldsymbol{\theta}_s, \mathcal{D}) p(\boldsymbol{\theta}_2 \mid \boldsymbol{\theta}_s, \mathcal{D})$, i.e., $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ are conditionally independent given $\boldsymbol{\theta}_s$ and the dataset $\mathcal{D}$. While this must be checked in applications, it is suitable for our setting, since we want both networks, $f_1(\mathcal{D}; \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$ and $f_2(\mathcal{D}; \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)$, to train independently well.

To derive our loss we study the components on the right of equation (1). We start with the integral in the denominator, for which we seek a closed form. Suppose we have trained the first model and seek to update the parameters of the second one while avoiding neural brainwashing. The following lemma provides an expression for the denominator of equation (1).

Lemma 2. *Suppose we have the maximum likelihood estimate* $(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$ *for the first model, write* $\mathrm{Card}(\boldsymbol{\theta}_1) + \mathrm{Card}(\boldsymbol{\theta}_s) = p_1 + p_s = p$*, and let the negative Hessian* $\boldsymbol{H}_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$ *of the log posterior probability distribution* $\log p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D})$ *evaluated at* $(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$ *be partitioned into four blocks corresponding to* $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$ *as*

$$\boldsymbol{H}_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) = \left[\begin{array}{c|c} \boldsymbol{H}_{11} & \boldsymbol{H}_{1s} \\ \hline \boldsymbol{H}_{s1} & \boldsymbol{H}_{ss} \end{array}\right].$$

If the parameters of each model follow Normal distributions, i.e., $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) \sim \mathcal{N}_p(\mathbf{0}, \sigma^2 \boldsymbol{I}_p)$*, with* $\boldsymbol{I}_p$ *the* p*-dimensional identity matrix, then*

$$\int p(\mathcal{D} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_s) p(\boldsymbol{\theta}_s, \boldsymbol{\theta}_1) d\boldsymbol{\theta}_1 = \exp\{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) - \frac{1}{2}\boldsymbol{v}^\top \boldsymbol{\Omega} \boldsymbol{v}\} \times (2\pi)^{p_1/2} |\det(\boldsymbol{H}_{11}^{-1})|^{1/2}, \quad (2)$$

where $\boldsymbol{v} = \boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s$ *and* $\boldsymbol{\Omega} = \boldsymbol{H}_{ss} - \boldsymbol{H}_{1s}^\top \boldsymbol{H}_{11}^{-1} \boldsymbol{H}_{1s}$ *.*

Proof. Provided in the appendix. □

Lemma 2 requires the maximum likelihood estimate $(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$, which can be hard to obtain with deep networks, since they have non-convex objective functions. In practice, one can train the network to convergence and treat the resulting parameters as maximum likelihood estimates. Our experiments show that the parameters obtained without optimizing to convergence can be used effectively. Moreover Haeffele & Vidal (2017) showed that networks relying on *positively homogeneous functions* have critical points that are either global minimizers or saddle points, and that training to convergence yields near-optimal solutions, which correspond to true maximum likelihood estimates.

Following Lemmas 1 and 2, as shown in the appendix,

$$\log p(\boldsymbol{\theta} \mid \mathcal{D}) \propto \log p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s) + \log p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s) + \log p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D}) + \frac{1}{2}\boldsymbol{v}^\top \boldsymbol{\Omega} \boldsymbol{v}. \quad (3)$$

To derive a loss function that prevents neural brainwashing, consider equation (3). The first term on its right-hand side corresponds to the log likelihood of the second model and can be replaced by the cross-entropy $\mathcal{L}_2(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s)$, and if we use a Gaussian prior on the parameters, the second term encodes an L^2 regularization. Since equation (3) depends only on the log likelihood of the second model $f_2(\mathcal{D}; \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)$, the information learned from the first model $f_1(\mathcal{D}; \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$ must reside in the conditional posterior probability $\log p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D})$, and the final term, $\frac{1}{2}\boldsymbol{v}^\top \boldsymbol{\Omega} \boldsymbol{v}$, must represent the interactions between the models $f_1(\mathcal{D}; \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)$ and $f_2(\mathcal{D}; \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$. Let us examine these.

The posterior probability $p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D})$ is intractable, so we apply a Laplace approximation (MacKay, 1992); we approximate the log posterior with a second-order Taylor expansion around the maximum likelihood estimate $(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$. This yields

$$\log p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D}) = \log p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s \mid \mathcal{D}) - \frac{1}{2}[(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)]^\top \boldsymbol{H}_p [(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)], \quad (4)$$

where $\boldsymbol{H}_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$ is the negative Hessian of the log posterior evaluated at the maximum likelihood estimate. The first derivative equals zero since it is evaluated at the maximum likelihood estimate.

Equation (4) yields a Gaussian approximation of the posterior with mean $(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$ and covariance matrix $\boldsymbol{H}_p^{-1}$, i.e.,

$$p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D}) \propto \exp\big\{ -\frac{1}{2}[(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)]^\top \boldsymbol{H}_p [(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)] \big\}. \quad (5)$$

As our parameter space is large it is impossible to compute the inverse of the negative Hessian $\boldsymbol{H}_p$, so we replace it with the diagonal of the Fisher information, $\mathrm{diag}(\boldsymbol{F})$. This approximation falsely presupposes that the parameters $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$ are independent, but it has already proven effective (Kirkpatrick et al., 2017; Pascanu & Bengio, 2014). One of its main advantages is that we can compute the Fisher information from the squared gradients, thereby avoiding any need for second derivatives.

Using equation (5) and the Fisher approximation we can express the log posterior as

$$\log p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D}) \propto \frac{\alpha}{2} \sum_{\theta_{s_i} \in \boldsymbol{\theta}_s} F_{\theta_{s_i}} (\theta_{s_i} - \hat{\theta}_{s_i})^2 \ , \tag{6}$$

where $F_{\theta_{s_i}}$ is the diagonal element corresponding to parameter θ_{s_i} in the diagonal approximation of the Fisher information matrix, which can be obtained from the trained model $f_1(\mathcal{D}; \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$.

Now consider the last term in equation (3), noting that $\boldsymbol{\Omega} = \boldsymbol{H}_{ss} - \boldsymbol{H}_{1s}^\top \boldsymbol{H}_{11}^{-1} \boldsymbol{H}_{1s}$, as defined in Lemma 2. As our previous approximation relies on the assumption of a diagonal Fisher information matrix, we have $\boldsymbol{H}_{1s} = \boldsymbol{0}$, leading to $\boldsymbol{\Omega} = \boldsymbol{H}_{ss}$, so

$$\frac{1}{2} \boldsymbol{v}^\top \boldsymbol{\Omega} \boldsymbol{v} = \frac{1}{2} \sum_{\theta_{s_i} \in \boldsymbol{\theta}_s} F_{\theta_{s_i}} (\theta_{s_i} - \hat{\theta}_{s_i})^2 \ . \tag{7}$$

The last two terms on the right-hand side of equation (3), as expressed in equation (6) and equation (7), can then be grouped. Combining the result with the first two terms, discussed below equation (3), yields our Weight Plasticity Loss,

$$\mathcal{L}_{\text{WPL}}(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s) = \mathcal{L}_2(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s) + \frac{\lambda}{2}(\|\boldsymbol{\theta}_s\|^2 + \|\boldsymbol{\theta}_2\|^2) + \frac{\alpha}{2} \sum_{\theta_{s_i} \in \boldsymbol{\theta}_s} F_{\theta_{s_i}} (\theta_{s_i} - \hat{\theta}_{s_i})^2, \tag{8}$$

where $F_{\theta_{s_i}}$ is the diagonal element corresponding to parameter θ_{s_i} in the Fisher information matrix obtained from the trained first model $f_1(\mathcal{D}; \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$. We omit the terms depending on $\boldsymbol{\theta}_1$ in equation (6) because we are optimizing with respect to $(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s)$ at this stage. The Fisher information in the last term encodes the importance of each shared weight for the first model's performance, so WPL encourages preserving any shared parameters that were important for the first model, while allowing others to undergo larger changes and thus to improve the accuracy of the second model.

3.1.1 Relation to Elastic weight consolidation

The final loss function obtained in equation (8) may look similar to that obtained by Kirkpatrick et al. (2017) when formulating their Elastic Weight Consolidation (EWC) to address catastrophic forgetting. However, the problem we address here is fundamentally different. Kirkpatrick et al. (2017) tackle sequential learning on different tasks, where a *single model* is sequentially trained using *two datasets*, and their goal is to maximize the posterior $p(\boldsymbol{\theta} \mid \mathcal{D}) = p(\boldsymbol{\theta} \mid \mathcal{D}_1, \mathcal{D}_2)$. By relying on Laplace approximations in neural networks (MacKay, 1992) and the connection between the Fisher information matrix and second-order derivatives (Pascanu & Bengio, 2014), EWC is then formulated as the loss $\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_B(\boldsymbol{\theta}) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta^\star_{A,i})^2$, where A and B refer to *two different tasks*, $\boldsymbol{\theta}$ encodes the network parameters and F_i is the Fisher information of θ_i.

Here we consider scenarios with a *single dataset* but *two models* with shared parameters, and aim to maximize the posterior $p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_s \mid \mathcal{D})$. The resulting WPL combines the original loss of the second model, a Fisher-weighted MSE term on the shared parameters and an L^2 regularizer on the parameters of the second model. Under mild assumptions we obtain a statistically-motivated loss function that is useful in practice. We believe this to be a valuable contribution in itself, but, more importantly, we show below that it can significantly reduce neural brainwashing.

3.2 WPL for Neural Architecture Search

In the previous section, we considered only two models being trained sequentially, but in practice one often seeks to train three or more models. Our approach is then unchanged, but each model shares parameters with several other models, which entails using diagonal approximations to Fisher information matrices for all previously-trained models from equation (3). In the remainder of this section, we discuss how our approach can be used for neural architecture search.

Consider using our WPL within the ENAS strategy of Pham et al. (2018). ENAS is a reinforcement-learning-based method that consists of two training processes: 1) sequentially train sampled models with shared parameters; and 2) train a controller RNN that generates model candidates. Incorporating our WPL within ENAS only affects 1).

The first step of ENAS consists of sampling a fixed number of architectures from the RNN controller, and training each architecture on B batches. This implies that our requirement for access to the maximum likelihood estimate of the previously-trained models is not satisfied, but we verify that in practice our WPL remains effective in this scenario. After sufficiently many epochs it is likely that all the parameters of a newly-sampled architecture are shared with previously-trained ones, and then we can consider that all parameters of new models are shared.

At the beginning of the search, the parameters of all models are randomly initialized. Adopting WPL directly from the start would therefore make it hard for the process to learn anything, as it would encourage some parameters to remain random. To better satisfy our assumption that the parameters of previously-trained models should be optimal, we follow the original ENAS training strategy for n epochs, with $n = 5$ for RNN search and $n = 3$ for CNN search in our experiments. We then incorporate our WPL and store the optimal parameters after each architecture is trained. We also update the Fisher information, which adds virtually no computational overhead, because $F_{\theta i} = (\partial \mathcal{L}/\partial \theta_i)^2$, where $\mathcal{L} = \sum_i \mathcal{L}_i$, with i indexing the previously-sampled architectures, and the derivatives are already computed for back-propagation. To ensure that these updates use the contributions from all previously-sampled architectures, we use a momentum-based update expressed as $F_{\theta_i}^t = (1-\eta)F_{\theta_i}^{t-1} + \eta(\partial \mathcal{L}/\partial \theta_i)^2$, with $\eta = 0.9$. Since such Fisher information is not computed at the MLE of the parameters, we flush the global Fisher buffer to zero every three epochs, yielding an increasingly accurate estimate of the Fisher information as optimization proceeds. We also use a scheduled decay for α in equation (8), which progressively gives us the ability to learn the parameters of the most promising architectures freely.

4 EXPERIMENTS

We first evaluate our weight plasticity loss (WPL) in the general scenario of training two models sequentially, both in the strict convergence case and when the weights of the first model are sub-optimal, and then evaluate the performance of our approach within the ENAS framework.

4.1 GENERAL SCENARIO: TRAINING TWO MODELS

To test WPL in the general scenario, we used the MNIST handwritten digit recognition dataset (LeCun & Cortes, 2010). We designed two feed-forward networks with 4 (Model A) and 6 (Model B) layers, respectively. All the layers of A are shared by B.

Let us first evaluate our approach in the strict convergence case. To this end, we trained A until convergence, thus obtaining a solution close to the MLE $\hat{\boldsymbol{\theta}}_A = (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$, since all our operations are positively homogeneous (Haeffele & Vidal, 2017). To compute the Fisher information, we used the backward gradients of $\boldsymbol{\theta}_s$ calculated on 200 images in the validation set. We then initialized $\boldsymbol{\theta}_s$ of Model B, $f_B(\mathcal{D}; (\boldsymbol{\theta}_2, \boldsymbol{\theta}_s))$, as $\hat{\boldsymbol{\theta}}_s$ and trained B by standard SGD with respect to all its parameters. Figure 2*(a)* compares the performance of training model B with and without our WPL. While without WPL the performance of A degrades as training B progresses, the use of our WPL allows us to maintain the initial performance of A, indicated as Baseline in the plot. This entails no loss of performance for B, whose final accuracy is virtually the same with and without WPL.

The assumption of optimal weights is usually hard to enforce. We therefore now turn to the more realistic loose convergence scenario. To evaluate the influence of sub-optimal weights for Model A on our approach, we trained Model A to different, increasingly lower, top 1 accuracies. As shown in Figure 2*(b)* and *(c)*, even in this setting our approach still significantly reduces neural brainwashing. We can quantify the relative reduction rate of such brainwashing as $d_A - d_{A+WPL}/d_A$, where $d = acc_A^* - acc$ is A's accuracy decay after training B. Our WPL can reduce brainwashing up to 99% for more converged model, and by 52% even for the loose case. This suggests that the Fisher information remains a reasonable empirical approximation to the weights' importance even when our optimality assumption is not satisfied. This suggests that our approach will generalize to applications, such as neural architecture search, as discussed below.

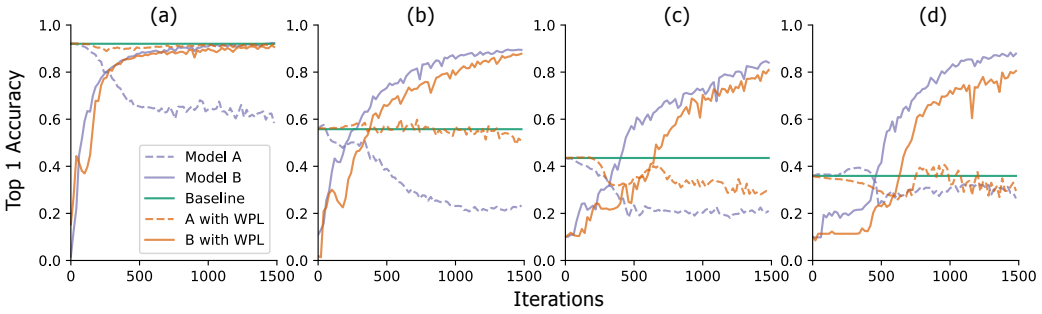

Figure 2: **From strict to loose convergence.** We conduct experiments on MNIST with tmodels, A and B, with shared parameters. We report the accuracy of Model A before training Model B (baseline, green) and the accuracy of Models A and B while training Model B with (orange) or without (blue) WPL. In *(a)* we show the results for strict convergence: A is initially trained to convergence. We then relax this assumption and train A to around 55% *(b)*, 43% *(c)*, and 38% *(d)* of its optimal accuracy. We see that our WPL is highly effective when A is trained to at least 40% of optimality; below, the Fisher information becomes too inaccurate to provide reliable importance weights. Thus WPL helps to reduce neural brainwashing, even when the weights are not optimal. WPL reduced brainwashing by up to 99.99% for *(a)* and *(b)*, and 5by up to 2% for *(c)*.

4.2 WPL for Neural architecture search

We now demonstrate the effectiveness of our WPL in a real-world application, neural architecture search. In particular, we incorporate it in the ENAS framework (Pham et al., 2018), which extensively relies on weight sharing across model candidates to speed up the search and thus, while effective, will suffer from neural brainwashing. To show this, we examine how the previously-trained architectures are affected by the training of new ones by evaluating the prediction error of each sampled architecture on a fraction of the validation dataset immediately after it is trained, denoted by err_1, and at the end of the epoch, denoted by err_2. A positive difference $err_2 - err_1$ for a specific architecture indicates that it has been brainwashed by others.

Following standard practice, we performed two experiments: RNN cell search on the PTB dataset and CNN micro-cell search on the CIFAR10 dataset. We report the mean error difference for all sampled architectures, the mean error difference for the 5 architectures with the lowest err_1, and the maximum error difference over all sampled architectures. Figure 3*(a)*, *(b)* and *(c)* plot these as functions of the training epochs for the RNN case, and similar plots for CNN search are in the appendix. The plots shows that without WPL the error differences are much larger than 0, clearly displaying the neural brainwashing effect. This is particularly pronounced in the first half of training, which can have a dramatic effect on the final results, since it corresponds to the phase where the algorithm searches for promising architectures. WPL significantly reduces the brainwashing, as shown by much lower error differences. The fact that, with WPL, these differences tend to decrease over time also shows that the observed Fisher information encodes an increasingly reliable notion of weight importance as training progresses. Owing to limited computational resources we estimate the Fisher information using only small validation batches, but use of larger batches could further improve our results by providing a more accurate estimate of the Fisher information.

In Figure 3*(d)*, we plot the average reward of all sampled architectures as a function of the training iterations. In the first half of training, the models trained with WPL tend to have lower rewards. This can be explained by the use of a large value for α in equation (8) during this phase; while such a large value may prevent the best models from achieving as high a reward as possible, it has the advantage of preventing the brainwashing of good models, and thus avoiding their being discarded early. This is shown by the fact that, in the second half of training, when we reduce α, the mean reward of the architectures trained with WPL is higher than without using it. In other words, our approach allows us to maintain better models until the end of training.

When the search is over, we train the best architecture from scratch and evaluate its final accuracy. Table 1 compares the results obtained without (ENAS) and with WPL (ENAS+WPL) with those

Table 1: **Results of the best models found.** We take the best model obtained during the search and train it from scratch. ENAS* corresponds to the results of Pham et al. (2018) obtained after extensive hyper-parameter search, while ENAS and ENAS+WPL were trained in comparable conditions. For both RNN and CNN search, our WPL gives a significant boost to ENAS, thus showing the importance of overcoming neural brainwashing. In the RNN case, our approach outperforms ENAS* without requiring extensive hyper-parameter tuning. The best results in each row are bold.

Datasets	Metric	ENAS*	ENAS	**ENAS + WPL**
PTB	perplexity	63.26	65.01	**61.9**
CIFAR10	top-1 error	**3.54**	4.87	3.81

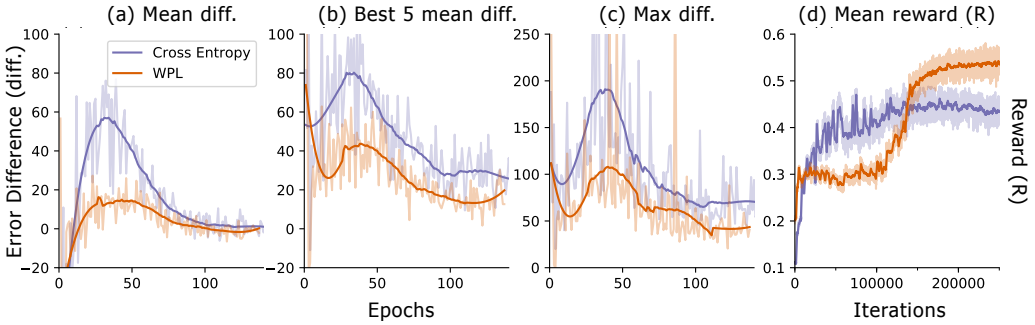

Figure 3: **Error difference during neural architecture search.** For each architecture, we compute the RNN error differences $err_2 - err_1$, where err_1 is the error right after training this architecture and err_2 the one after all architectures are trained in the current epoch. We plot (a) the mean difference over all sampled models, (b) the mean difference over the 5 models with lowest err_1, and (c) the max difference over all models. The plots clearly evidence that WPL reduces neural brainwashing; the error differences are much closer to 0. Quantitatively, the brainwashing reduction can achieve upto 95% for *(a)*, 59% for *(b)*and 51% for *(c)*. In (d), we plot the average reward of the sampled architectures as a function of training iterations. Note that, while WPL initially leads to lower rewards, due to a large weight α in equation (8), by reducing the brainwashing, it allows the controller to later sample better architectures, as indicated by the higher reward in the second half.

from the original ENAS paper (ENAS*), which were obtained after conducting an extensive hyper-parameter search. For both datasets, using WPL improves final model accuracy, thus showing the importance of overcoming neural brainwashing. In the case of PTB, our approach even outperforms ENAS*, without extensive hyper-parameter tuning. Based on the gap between ENAS and ENAS*, we anticipate that such a tuning procedure could further boost our results. In any event, we believe that these results already clearly show the benefits of reducing neural brainwashing in applications.

5 Conclusion

This paper has identified *neural brainwashing* in the context of multi-model sequential training: the shared weights of previously-trained models are overwritten during training of subsequent models, leading to performance degradation. We show that the degree of degradation is linked to the proportion of shared weights, and introduce a statistically-motivated weight plasticity loss (WPL) to overcome this. Our experiments on multi-model training and on neural architecture search clearly show the effectiveness of WPL in reducing brainwashing and yielding better architectures, leading to improved results in both natural language processing and computer vision tasks. We believe that the impact of WPL goes beyond the tasks studied in this paper. In future work, we plan to integrate WPL within other neural architecture search strategies in which weight sharing occurs and to study its use in other multi-model contexts, such as for ensemble learning.

REFERENCES

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. *The European Conference on Computer Vision (ECCV)*, 2018.

Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Inc., 1996.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *International Conference on Learning Representations (ICLR), Conference track*, 2017.

Kamil Bennani-Smires, Claudiu Musat, Andreea Hossmann, and Michael Baeriswyl. Gitgraph - from computational subgraphs to smaller architecture search spaces. *International Conference on Learning Representations (ICLR), Workshop track*, 2018.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. *AAAI*, 2018.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

Tianqi Chen, Ian J. Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *International Conference on Learning Representations (ICLR), Conference track*, 2016.

Thomas G. Dietterich. Ensemble methods in machine learning. *Multiple Classifier Systems*, pp. 1–15, 2000.

T. Elsken, J. Hendrik Metzen, and F. Hutter. Neural Architecture Search: A Survey. *arXiv preprint arXiv:1808.05377*, 2018.

David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, pp. 69–93, 1991.

Benjamin D. Haeffele and René Vidal. Global optimality in neural network training. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4390–4398, 2017.

Xu He and Herbert Jaeger. Overcoming catastrophic interference by conceptors. *arXiv preprint arXiv:1707.04853*, 2017.

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.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2009.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

Zhizhong Li and Derek Hoiem. Learning without forgetting. pp. 614–629, 2016.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *International Conference on Learning Representations (ICLR), Conference track*, 2018a.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

David J. C. MacKay. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472, 1992.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, pp. 114–119. Association for Computational Linguistics, 1994.

R. Negrinho and G. Gordon. DeepArchitect: Automatically Designing and Training Deep Architectures. *arXiv preprint arXiv:1704.08792*, 2017.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, 2010.

Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *International Conference on Learning Representations (ICLR), Conference track*, 2014.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. *International Conference on Machine Learning (ICML)*, 2018.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. *International Conference on Machine Learning (ICML)*, 2017.

Jorma Rissanen. Fisher information and stochastic complexity. *IEEE Trans. Information Theory*, 42, 1996.

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.

Daniel Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. 2013.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

L. Xie and A. Yuille. Genetic cnn. *IEEE International Conference on Computer Vision (ICCV)*, 2017.

Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *arXiv preprint arXiv:1708.02709*, 2017.

Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. *International Conference on Learning Representations (ICLR), Conference track*, 2017.

A PROOFS

Lemma 1. *Given a dataset $\mathcal{D}$ and two architectures with shared parameters $\boldsymbol{\theta}_s$ and private parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, and provided that $p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \mid \boldsymbol{\theta}_s, \mathcal{D}) = p(\boldsymbol{\theta}_1 \mid \boldsymbol{\theta}_s, \mathcal{D})p(\boldsymbol{\theta}_2 \mid \boldsymbol{\theta}_s, \mathcal{D})$, we have*

$$p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_s \mid \mathcal{D}) \propto \frac{p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s)}{\int p(\mathcal{D} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)d\boldsymbol{\theta}_1}. \tag{1}$$

Proof. Using Bayes' theorem and ignoring constants, we have

$$\begin{aligned}
p(\boldsymbol{\theta} \mid \mathcal{D}) &= \frac{p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_s, \mathcal{D})}{p(\mathcal{D})} \\
&\propto p(\boldsymbol{\theta}_1 \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s, \mathcal{D})p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s, \mathcal{D}) \\
&= p(\boldsymbol{\theta}_1 \mid \boldsymbol{\theta}_s, \mathcal{D})p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s) \\
&\propto \frac{p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s, \mathcal{D})p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s)}{p(\mathcal{D}, \boldsymbol{\theta}_s)} \\
&\propto \frac{p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s, \mathcal{D})p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s)}{\int p(\mathcal{D} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s, \boldsymbol{\theta}_1)d\boldsymbol{\theta}_1} \\
&\propto \frac{p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D})p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s)}{\int p(\mathcal{D} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s, \boldsymbol{\theta}_1)d\boldsymbol{\theta}_1},
\end{aligned}$$

where we used the conditional independence assumption $p(\boldsymbol{\theta}_1 \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s, \mathcal{D}) = p(\boldsymbol{\theta}_1 \mid \boldsymbol{\theta}_s, \mathcal{D})$ in the third line. □

We now derive a closed-form expression for the denominator of equation (1).

Lemma 2. *Suppose we have the maximum likelihood estimate $(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$ for the first model, write $\mathrm{Card}(\boldsymbol{\theta}_1) + \mathrm{Card}(\boldsymbol{\theta}_s) = p_1 + p_s = p$, and let the negative Hessian $\boldsymbol{H}_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$ of the log posterior probability distribution $\log p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D})$ evaluated at $(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$ be partitioned into four blocks corresponding to $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$ as*

$$\boldsymbol{H}_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) = \left[\begin{array}{c|c} \boldsymbol{H}_{11} & \boldsymbol{H}_{1s} \\ \hline \boldsymbol{H}_{s1} & \boldsymbol{H}_{ss} \end{array}\right].$$

If the parameters of each model follow Normal distributions, i.e., $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) \sim \mathcal{N}_p(\mathbf{0}, \sigma^2 \boldsymbol{I}_p)$, with $\boldsymbol{I}_p$ the p-dimensional identity matrix, then

$$\int p(\mathcal{D} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s, \boldsymbol{\theta}_1)d\boldsymbol{\theta}_1 = \exp\{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) - \frac{1}{2}\boldsymbol{v}^\top \boldsymbol{\Omega} \boldsymbol{v}\} \times (2\pi)^{p_1/2} |\det(\boldsymbol{H}_{11}^{-1})|^{1/2}, \tag{2}$$

where $\boldsymbol{v} = \boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s$ and $\boldsymbol{\Omega} = \boldsymbol{H}_{ss} - \boldsymbol{H}_{1s}^\top \boldsymbol{H}_{11}^{-1} \boldsymbol{H}_{1s}$.

Proof. We have

$$p(\mathcal{D} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s, \boldsymbol{\theta}_1) \propto e^{l(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)^T(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)/2\sigma^2} = e^{l_p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)},$$

where $l(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) = \log p(\mathcal{D} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$, and $l_p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) = l(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)^T(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)/2\sigma^2$.

Let $\boldsymbol{H}_p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) = \boldsymbol{H}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) + \sigma^{-2}\boldsymbol{I}_p$ be the negative Hessian of $l_p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$, with $\boldsymbol{I}_p$ the p-dimensional identity matrix and $\boldsymbol{H}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$ the negative Hessian of $l(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$.

Using the second-order Taylor expansion of $l_p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)$ around its maximum likelihood estimate $(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)$, we have

$$l_p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) = l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) - \frac{1}{2}[(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)]^T \boldsymbol{H}_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)[(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)]; \tag{9}$$

the first derivative is zero since it is evaluated at the maximum likelihood estimate. We now partition our negative Hessian matrix as

$$\boldsymbol{H}_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) = \left[\begin{array}{c|c} \boldsymbol{H}_{11} & \boldsymbol{H}_{1s} \\ \hline \boldsymbol{H}_{s1} & \boldsymbol{H}_{ss} \end{array}\right],$$

which gives

$$\begin{aligned}\boldsymbol{A} &= [(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)]^T \boldsymbol{H}_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)[(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) - (\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)] \\ &= (\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1)^T \boldsymbol{H}_{11}(\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1) + (\boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s)^T \boldsymbol{H}_{ss}(\boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s) + (\boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s)^T \boldsymbol{H}_{s1}(\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1) + (\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1)^T \boldsymbol{H}_{1s}(\boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s) \\ &= (\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1)^T \boldsymbol{H}_{11}(\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1) + (\boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s)^T \boldsymbol{H}_{ss}(\boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s) + (\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1)^T (\boldsymbol{H}_{1s} + \boldsymbol{H}_{s1}^T)(\boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s).\end{aligned}$$

Let us define $\boldsymbol{u} = \boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1$, $\boldsymbol{v} = \boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s$ and $\boldsymbol{w} = \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}\boldsymbol{v}$. We then have

$$\begin{aligned}(\boldsymbol{u} + \boldsymbol{w})^T \boldsymbol{H}_{11}(\boldsymbol{u} + \boldsymbol{w}) &= \boldsymbol{u}^T \boldsymbol{H}_{11}\boldsymbol{u} + \boldsymbol{u}^T \boldsymbol{H}_{11}\boldsymbol{w} + \boldsymbol{w}^T \boldsymbol{H}_{11}\boldsymbol{w} + \boldsymbol{w}^T \boldsymbol{H}_{11}\boldsymbol{u} \\ &= (\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1)^T \boldsymbol{H}_{11}(\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1) + (\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1)^T \boldsymbol{H}_{11}\boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}(\boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s) \\ &\qquad + \boldsymbol{v}^T \boldsymbol{H}_{1s}^T \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{11}\boldsymbol{H}_{1s}\boldsymbol{v} + \boldsymbol{v}^T \boldsymbol{H}_{1s}^T \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{11}(\boldsymbol{\theta}_1 - \hat{\boldsymbol{\theta}}_1) \\ &= \boldsymbol{A} - \boldsymbol{v}^T \boldsymbol{H}_{ss}\boldsymbol{v} + \boldsymbol{v}^T \boldsymbol{H}_{1s}^T H_{11}^{-1}\boldsymbol{H}_{1s}\boldsymbol{v} \\ &= \boldsymbol{A} - \boldsymbol{v}^T(\boldsymbol{H}_{ss} - \boldsymbol{H}_{1s}^T \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s})v \\ &= \boldsymbol{A} - \boldsymbol{v}^T \boldsymbol{\Omega}\boldsymbol{v},\end{aligned}$$

with $\boldsymbol{\Omega} = \boldsymbol{H}_{ss} - \boldsymbol{H}_{1s}^T \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}$.

Thus

$$\boldsymbol{A} = (\boldsymbol{u} + \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}v)^T \boldsymbol{H}_{11}(\boldsymbol{u} + \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}\boldsymbol{v}) + \boldsymbol{v}^T \boldsymbol{\Omega}\boldsymbol{v}. \tag{10}$$

Given equation (10), we are now able to prove Lemma 2, as

$$\begin{aligned}\int e^{l_p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s)} d\boldsymbol{\theta}_1 &= \int e^{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) - \frac{1}{2}\boldsymbol{A}} d\boldsymbol{\theta}_1 \\ &= \int e^{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)} e^{-\frac{1}{2}\boldsymbol{A}} d\boldsymbol{\theta}_1 \\ &= e^{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)} \int e^{-\frac{1}{2}\boldsymbol{A}} d\boldsymbol{\theta}_1 \\ &= e^{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)} \int e^{-\frac{1}{2}((\boldsymbol{u} + \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}\boldsymbol{v})^T \boldsymbol{H}_{11}(\boldsymbol{u} + \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}\boldsymbol{v}) + \boldsymbol{v}^T \boldsymbol{\Omega}\boldsymbol{v})} d\boldsymbol{\theta}_1 \\ &= e^{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s)} \int e^{-\frac{1}{2}((\boldsymbol{u} + \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}\boldsymbol{v})^T \boldsymbol{H}_{11}(\boldsymbol{u} + \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}\boldsymbol{v}))} e^{-\frac{1}{2}\boldsymbol{v}^T \boldsymbol{\Omega}\boldsymbol{v}} d\boldsymbol{\theta}_1 \\ &= e^{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) - \frac{1}{2}\boldsymbol{v}^T \boldsymbol{\Omega}\boldsymbol{v}} \int e^{-\frac{1}{2}(\boldsymbol{\theta}_1 - \boldsymbol{z})^T \boldsymbol{H}_{11}(\boldsymbol{\theta}_1 - \boldsymbol{z})} d\boldsymbol{\theta}_1 \\ &= e^{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) - \frac{1}{2}\boldsymbol{v}^T \boldsymbol{\Omega}\boldsymbol{v}} (2\pi)^{\frac{p_1}{2}} |\det(\boldsymbol{H}_{11}^{-1})|^{\frac{1}{2}} (2\pi)^{-\frac{p_1}{2}} |\det(\boldsymbol{H}_{11}^{-1})|^{-\frac{1}{2}} \int e^{-\frac{1}{2}(\boldsymbol{\theta}_1 - \boldsymbol{z})^T \boldsymbol{H}_{11}(\boldsymbol{\theta}_1 - \boldsymbol{z})} d\boldsymbol{\theta}_1 \\ &= e^{l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) - \frac{1}{2}\boldsymbol{v}^T \boldsymbol{\Omega}\boldsymbol{v}} (2\pi)^{\frac{p_1}{2}} |\det(\boldsymbol{H}_{11}^{-1})|^{\frac{1}{2}},\end{aligned}$$

where we re-arranged the terms so that the integral is over a normal distribution with mean $\boldsymbol{z} = \hat{\boldsymbol{\theta}}_1 - \boldsymbol{H}_{11}^{-1}\boldsymbol{H}_{1s}(\boldsymbol{\theta}_s - \hat{\boldsymbol{\theta}}_s)$ and covariance matrix $\boldsymbol{H}_{11}^{-1}$, which can be computed in closed form. □

From Lemma 1 and Lemma 2, we can obtain equation (3) by replacing the denominator with the closed form above and taking the log on both size of equation (1). This yields

$$\begin{aligned}\log p(\boldsymbol{\theta}|\mathcal{D}) &\propto \log p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s) + \log p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) + \log p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s) - \log\{\int p(\mathcal{D} \mid \boldsymbol{\theta}_1, \boldsymbol{\theta}_s) p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) d\boldsymbol{\theta}_1\} \\ &= \log p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s) + \log p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s) + \log p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s) - l_p(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_s) + \frac{1}{2}\boldsymbol{v}^T \boldsymbol{\Omega}\boldsymbol{v} \\ &\propto \log p(\mathcal{D} \mid \boldsymbol{\theta}_2, \boldsymbol{\theta}_s) + \log p(\boldsymbol{\theta}_2, \boldsymbol{\theta}_s) + \log p(\boldsymbol{\theta}_1, \boldsymbol{\theta}_s \mid \mathcal{D}) + \frac{1}{2}\boldsymbol{v}^T \boldsymbol{\Omega}\boldsymbol{v}\ .\end{aligned}$$

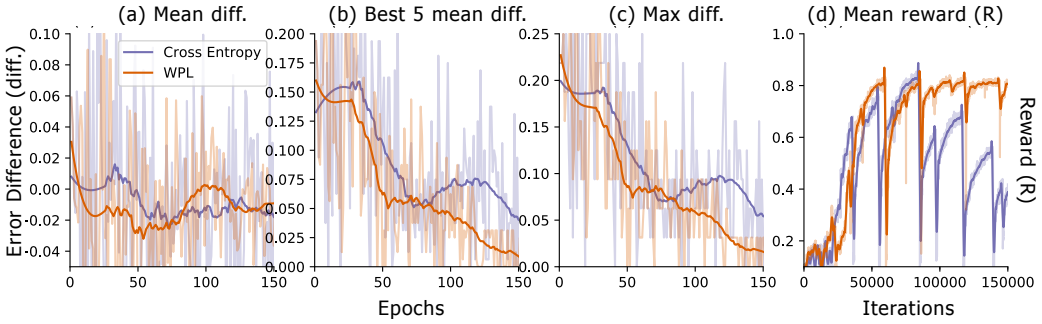

Figure 4: **Error differences when searching for CNN architectures.** Quantitatively, the brain-washing effect is reduced by up to 99% for *(a)*, 96% for *(b)*, and 98% for *(c)*.

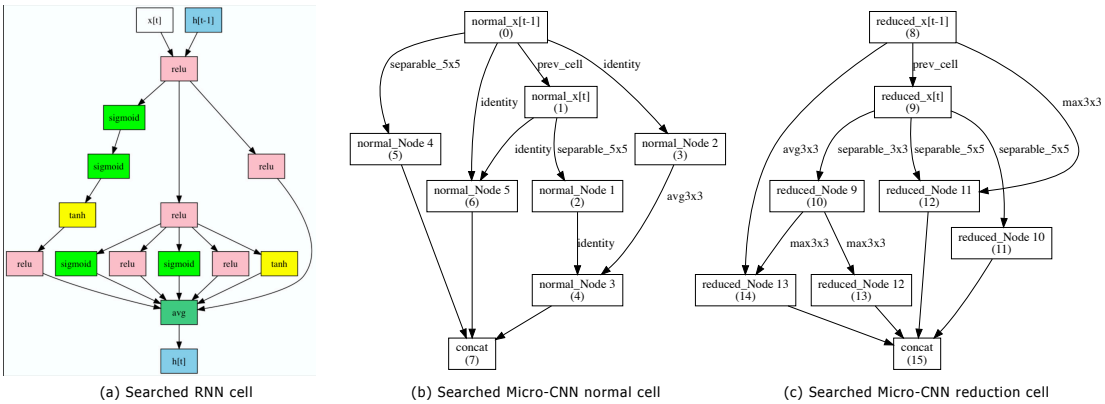

Figure 5: **Best architectures found for RNN and CNN.** We display the best architecture found by ENAS+WPL, in *(a)* for the RNN cell, and in *(b)* and *(c)* for the CNN normal and reduction cells.

B PLOTS FOR CNN SEARCH

In our CNN search experiment, we search for a"micro" cell as in (Pham et al., 2018). We employ the hyper-parameters available in the released ENAS code. The plots depicting error difference as a function of training epochs as provided in Figure 4 *(a)*, *(b)*and *(c)*. Note that here again the original ENAS is subject to neural brainwashing, and our WPL helps reducing it. In Figure 4 *(d)*, we show the mean reward as training progresses. While the shape of the reward curve is different from the RNN case, because of a different formulation of the reward function, the general trend is the same; Our approach initially produces lower rewards, but is better at maintaining good models until the end of the search, as indicated by higher rewards in the second half of training.

C BEST ARCHITECTURES FOUND BY THE SEARCH

In Figure 5, we show the best architectures found by our neural architecture search for the RNN and CNN cases.