Appendix

This document contains the materials supporting and extending the discussion in the main text. It is organised as follows:

- **Appendix A** collects the proofs to all the Theorems and Corollaries of the main text. Additionally, we elaborate further observations derived from our theoretical framework.
- **Appendix B** provides all the details of our experimental setup.
- **Appendix C** gathers additional results supporting the experiments described in the main text.

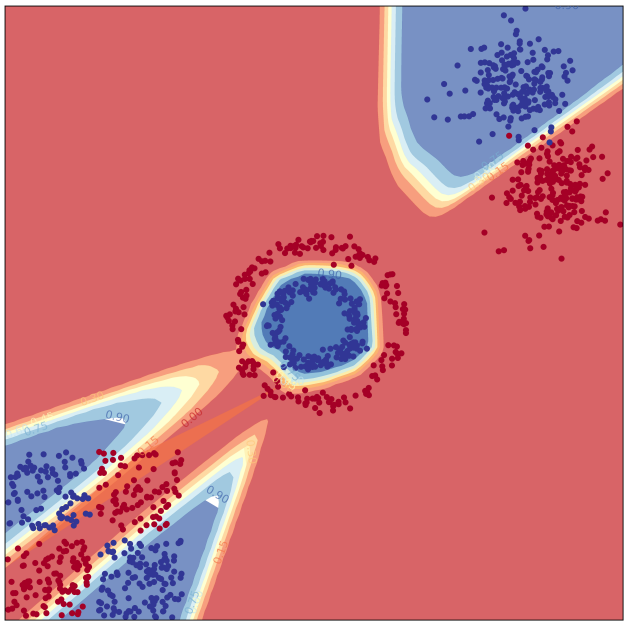

Figure 4: The multitask solution for the toy challenge in Figure 1. The same network has been used. This demonstrates that a multitask solution exists, despite standard optimization algorithms are not able to find it.

A PROOFS AND FURTHER DISCUSSION

A.0.1 NOTATION AND FORMALISM

First, we shortly recap the notation and formalism introduced in our discussion.

We consider a sequence of supervised learning tasks $\mathcal{T}_1, \dots, \mathcal{T}_T$, each associated with a dataset $D_t = \{(x_1, y_1), \dots, (x_{n_t}, y_{n_t})\}$, where $(x, y) \in \mathcal{X} \times \mathcal{Y}_t$. We describe a neural network by its set of parameters $\Theta_t \subseteq \mathbb{R}^P$, P denoting the total number of parameters in the network. We tipically use the index t to refer to the task currently being learned and o to a general previous task.

We use bold to indicate a vector or a matrix, e.g. $\mathbf{0}$ is the null vector. $\boldsymbol{\theta}$ refers to a general value taken by the parameters, and $\boldsymbol{\theta}_t$ is the solution found on task $\mathcal{T}_t$. Moreover, we write $\boldsymbol{\Delta}_t := \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$, i.e. the update to the parameters following task $\mathcal{T}_t$. Simply, $\boldsymbol{\Delta}_0 = \mathbf{0}$, $\boldsymbol{\theta}_0$ being the initialisation point.

We use $l_t(x, y, \boldsymbol{\theta})$ to denote the t-th task loss function, and write the average loss over the dataset as $\mathcal{L}_t(\boldsymbol{\theta})$, i.e. $\mathcal{L}_t(\boldsymbol{\theta}) = \frac{1}{n_t} \sum_{i=1}^{n_t} l_t(x_i, y_i, \boldsymbol{\theta})$. We shorten the loss measured at the end of training $\mathcal{L}_t(\boldsymbol{\theta}_t)$ to $\mathcal{L}_t^\star$.

The vector of first-order derivatives or gradient of the loss with respect to the parameters $\boldsymbol{\theta}$ is $\boldsymbol{\nabla}\mathcal{L}_t(\boldsymbol{\theta}) = \frac{\partial \mathcal{L}_t(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ and the matrix of second-order-derivativatives or Hessian of the loss with respect to the parameters $\boldsymbol{\theta}$ is $\mathbf{H}_t(\boldsymbol{\theta}) = \frac{\partial^2 \mathcal{L}_t(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2}$. In the interest of space, we shorten the Hessian evaluated at the end of training $\mathbf{H}_t(\boldsymbol{\theta}_t)$ to $\mathbf{H}_t^\star$.

In our discussion we use w.l.o.g. $\Theta_t \subseteq \Theta$. We will now justify this choice. Consider the case in which the network is somewhat different for each task (e.g. in the case of Progressive Networks (Rusu et al., 2016) a new component is added to the network for each task). We then write $\Theta_i \neq \Theta_j$ for any $i, j \in [T]$. Define $\Theta = \bigcup_{t=1}^{T} \Theta_t$ to be the *final parameter space*. Accordingly, each task update $\boldsymbol{\Delta}_t$ belongs to a subspace of the final parameter space, i.e. $\boldsymbol{\theta}_t \in \Theta_t \subseteq \Theta$. Our analysis applies to a general Θ, thus carrying over to any choice of Θ_t. Intuitively, using new parameters for each task introduces orthogonality between the parameters (and correspondingly their update vectors) over task-specific dimensions in Θ. In order to be comprehensive, we use a general Θ in our analysis.

A.1 OMITTED PROOFS

A.1.1 APPROXIMATIONS OF FORGETTING

In the following, we show the steps leading to Equation 3.

We start from the Taylor expansion (Equation 2) of the loss around $\boldsymbol{\theta}_t$:

$$\mathcal{E}_o(\boldsymbol{\theta}_t) = (\boldsymbol{\theta}_t - \boldsymbol{\theta}_o)^\intercal \boldsymbol{\nabla}\mathcal{L}_o(\boldsymbol{\theta}_o) + \frac{1}{2}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_o)^\intercal \mathbf{H}_o^\star (\boldsymbol{\theta}_t - \boldsymbol{\theta}_o) + \boldsymbol{O}(\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_o\|^3)$$

Using Assumptions 1-2 we write:

$$\begin{aligned}
\mathcal{E}_o(t) &= \frac{1}{2}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_o)^\intercal \mathbf{H}_o^\star (\boldsymbol{\theta}_t - \boldsymbol{\theta}_o) + \mathcal{O}(\delta \cdot \epsilon) \\
&= \frac{1}{2}\left(\sum_{\tau=o+1}^{t} \boldsymbol{\Delta}_\tau\right)^\intercal \mathbf{H}_o^\star \left(\sum_{\tau=o+1}^{t} \boldsymbol{\Delta}_\tau\right) + \mathcal{O}(\delta \cdot \epsilon) \\
&= \underbrace{\frac{1}{2}\left(\sum_{\tau=o+1}^{t-1} \boldsymbol{\Delta}_\tau\right)^\intercal \mathbf{H}_o^\star \left(\sum_{\tau=o+1}^{t-1} \boldsymbol{\Delta}_\tau\right)}_{\mathcal{E}_o(t-1)} + \frac{1}{2}\boldsymbol{\Delta}_t^\intercal \mathbf{H}_o^\star \boldsymbol{\Delta}_t + \\
&\quad + \frac{1}{2}\left(\sum_{\tau=o+1}^{t-1} \boldsymbol{\Delta}_\tau\right)^\intercal \mathbf{H}_o^\star \boldsymbol{\Delta}_t + \frac{1}{2}\boldsymbol{\Delta}_t^\intercal \mathbf{H}_o^\star \left(\sum_{\tau=o+1}^{t-1} \boldsymbol{\Delta}_\tau\right) + \mathcal{O}(\delta \cdot \epsilon) \\
&= \mathcal{E}_o(t-1) + \frac{1}{2}\boldsymbol{\Delta}_t^\intercal \mathbf{H}_o^\star \boldsymbol{\Delta}_t + \left(\sum_{\tau=o+1}^{t-1} \boldsymbol{\Delta}_\tau\right)^\intercal \mathbf{H}_o^\star \boldsymbol{\Delta}_t + \mathcal{O}(\delta \cdot \epsilon)
\end{aligned}$$

Based on this formulation, we characterize the average forgetting as follows:

$$
\begin{aligned}
\mathcal{E}(t) &= \frac{1}{t}\sum_{o=1}^{t}\mathcal{E}_o(t) = \frac{1}{t}\underbrace{\mathcal{E}_t(t)}_{=0} + \frac{1}{t}\sum_{o=1}^{t-1}\mathcal{E}_o(t) \\
&= \frac{1}{t}\sum_{o=1}^{t-1}\left[\mathcal{E}_o(t-1) + \frac{1}{2}\boldsymbol{\Delta}_t^\intercal \mathbf{H}_o^\star \boldsymbol{\Delta}_t + \left(\sum_{\tau=o+1}^{t-1}\boldsymbol{\Delta}_\tau\right)^\intercal \mathbf{H}_o^\star \boldsymbol{\Delta}_t + \mathcal{O}(\delta\cdot\epsilon)\right] \\
&= \frac{1}{t}\sum_{o=1}^{t-1}[\mathcal{E}_o(t-1)] + \frac{1}{t}\sum_{o=1}^{t-1}\left[\frac{1}{2}\boldsymbol{\Delta}_t^\intercal \mathbf{H}_o^\star \boldsymbol{\Delta}_t\right] + \frac{1}{t}\sum_{o=1}^{t-1}\left[\left(\sum_{\tau=o+1}^{t-1}\boldsymbol{\Delta}_\tau\right)^\intercal \mathbf{H}_o^\star \boldsymbol{\Delta}_t\right] + \mathcal{O}(\delta\cdot\epsilon) \\
&= \frac{t-1}{t}\cdot\mathcal{E}(t-1) + \frac{1}{2t}\boldsymbol{\Delta}_t^\intercal\left[\sum_{o=1}^{t-1}\mathbf{H}_o^\star\right]\boldsymbol{\Delta}_t + \frac{1}{t}\left[\underbrace{\sum_{o=1}^{t-1}(\boldsymbol{\theta}_{t-1}-\boldsymbol{\theta}_o)^\intercal \mathbf{H}_o^\star}_{\boldsymbol{v}^\intercal}\right]\boldsymbol{\Delta}_t + \mathcal{O}(\delta\cdot\epsilon) \\
&= \frac{1}{t}\left((t-1)\cdot\mathcal{E}(t-1) + \frac{1}{2}\boldsymbol{\Delta}_t^\intercal(\sum_{o=1}^{t-1}\mathbf{H}_o^\star)\boldsymbol{\Delta}_t + \boldsymbol{v}^\intercal\boldsymbol{\Delta}_t\right) + \mathcal{O}(\delta\cdot\epsilon)
\end{aligned}
$$

A.1.2 THEOREM 1

We want to prove the following statement.

Theorem 1 (Null forgetting). For any continual learning algorithm satisfying Assumptions 1-2 the following relationship between previous and current forgetting exists:

$$
\mathcal{E}(1),\ldots,\mathcal{E}(t-1)=0 \implies \mathcal{E}(t) = \frac{1}{2}\boldsymbol{\Delta}_t^\intercal\left(\frac{1}{t}\cdot\sum_{o=1}^{t-1}\mathbf{H}_o^\star\right)\boldsymbol{\Delta}_t
$$

Proof. We prove the above statement by induction. We prove the base case for $t=1$ and $t=2$, since some terms trivially cancel out for $t=1$.

Base case 1: $\mathcal{E}(1)=0 \implies \mathcal{E}(2)=\frac{1}{2}\boldsymbol{\Delta}_2^\intercal \boldsymbol{H}_1^\star \boldsymbol{\Delta}_2$. Notice that by definition, $\mathcal{E}(1)=\mathcal{E}_1(1)=0$.

Using Equation 3 we can write $\mathcal{E}(2)$:

$$
\begin{aligned}
\mathcal{E}(2) &= \mathcal{E}(1) + \frac{1}{2}\boldsymbol{\Delta}_2^\intercal(\boldsymbol{H}_1^\star)\boldsymbol{\Delta}_2 + (\boldsymbol{\theta}_1-\boldsymbol{\theta}_1)^\intercal \boldsymbol{H}_1^\star\boldsymbol{\Delta}_2 \\
&= 0 + \frac{1}{2}\boldsymbol{\Delta}_2^\intercal \boldsymbol{H}_1^\star\boldsymbol{\Delta}_2 + 0
\end{aligned}
$$

Base case 2: $\mathcal{E}(1)=0, \mathcal{E}(2)=0 \implies \mathcal{E}(3)=\frac{1}{2}\boldsymbol{\Delta}_3^\intercal(\frac{1}{3}\boldsymbol{H}_1^\star+\frac{1}{3}\boldsymbol{H}_2^\star)\boldsymbol{\Delta}_3$.
Using the last result from case 1, we have that :

$$
\mathcal{E}(2) = \frac{1}{2}\boldsymbol{\Delta}_2^\intercal \boldsymbol{H}_1^\star\boldsymbol{\Delta}_2 = 0
$$

The latter equation implies that $\boldsymbol{\Delta}_2^\intercal \boldsymbol{H}_1^1 = \mathbf{0}$. Plugging it into the value of $\mathcal{E}(3)$ given by Equation 3:

$$
\begin{aligned}
\mathcal{E}(3) &= \frac{1}{3}\left(2\cdot\mathcal{E}(2) + \frac{1}{2}\boldsymbol{\Delta}_3^\intercal(\boldsymbol{H}_2^\star+\boldsymbol{H}_1^\star)\boldsymbol{\Delta}_3 + \sum_{t=1}^{2}(\boldsymbol{\theta}_2-\boldsymbol{\theta}_t)^\intercal \boldsymbol{H}_t^\star\boldsymbol{\Delta}_3\right) \\
&= 0 + \frac{1}{2}\boldsymbol{\Delta}_3^\intercal(\frac{1}{3}\boldsymbol{H}_2^\star+\frac{1}{3}\boldsymbol{H}_1^\star)\boldsymbol{\Delta}_3 + \frac{1}{3}\underbrace{\boldsymbol{\Delta}_2^\intercal \boldsymbol{H}_1^1}_{=\mathbf{0}}\boldsymbol{\Delta}_3 + \frac{1}{3}\underbrace{(\boldsymbol{\theta}_2-\boldsymbol{\theta}_2)^\intercal}_{=\mathbf{0}}\boldsymbol{H}_t^\star\boldsymbol{\Delta}_3
\end{aligned}
$$

Induction step: if $\mathcal{E}(\tau) = 0 \ \forall \tau < t$ *then* $\mathcal{E}(t) \geq 0$.
We start by writing out $\mathcal{E}(t)$.

$$\mathcal{E}(t) = \frac{1}{t}\Bigg((t-1)\cdot\mathcal{E}(t-1) + \frac{1}{2}\boldsymbol{\Delta}_t^\intercal\big(\sum_{\tau=1}^{t-1}\boldsymbol{H}_\tau^\star\big)\boldsymbol{\Delta}_t + \sum_{\tau=1}^{t-1}(\boldsymbol{\theta}_{t-1}-\boldsymbol{\theta}_\tau)^\intercal\boldsymbol{H}_\tau^\star\,\boldsymbol{\Delta}_t\Bigg)$$
$$= 0 + \frac{1}{2}\boldsymbol{\Delta}_t^\intercal\big(\sum_{\tau=1}^{t-1}\boldsymbol{H}_\tau^\star\big)\boldsymbol{\Delta}_t + \sum_{\tau=1}^{t-1}(\boldsymbol{\Delta}_{\tau+1}+\cdots+\boldsymbol{\Delta}_{t-1})^\intercal\boldsymbol{H}_\tau^\star\,\boldsymbol{\Delta}_t$$

The induction step is accomplished if $\sum_{\tau=1}^{t-1}(\boldsymbol{\Delta}_{\tau+1}+\cdots+\boldsymbol{\Delta}_{t-1})^\intercal\boldsymbol{H}_\tau^\star = \sum_{\tau=1}^{t-2}(\boldsymbol{\Delta}_{\tau+1}+\cdots+\boldsymbol{\Delta}_{t-1})^\intercal\boldsymbol{H}_\tau^\star = \mathbf{0}$. Consider now the case in which $\sum_{\tau=1}^{t-1}(\boldsymbol{\Delta}_{\tau+1}+\cdots+\boldsymbol{\Delta}_{t-1})^\intercal\boldsymbol{H}_\tau^\star \neq \mathbf{0}$. It follows that:

$$\sum_{\tau=1}^{t-2}(\boldsymbol{\Delta}_{\tau+1}+\cdots+\boldsymbol{\Delta}_{t-1})^\intercal\boldsymbol{H}_\tau^\star \neq \mathbf{0}$$
$$\sum_{\tau=1}^{t-2}\boldsymbol{\Delta}_{t-1}^\intercal\boldsymbol{H}_\tau^\star + \sum_{\tau=1}^{t-2}(\boldsymbol{\Delta}_{\tau+1}+\cdots+\boldsymbol{\Delta}_{t-2})^\intercal\boldsymbol{H}_\tau^\star \neq \mathbf{0}$$

Multilying by $\boldsymbol{\Delta}_{t-1}$ on the right we get:

$$\sum_{\tau=1}^{t-2}\boldsymbol{\Delta}_{t-1}^\intercal\boldsymbol{H}_\tau^\tau\boldsymbol{\Delta}_{t-1} + \sum_{\tau=1}^{t-3}(\boldsymbol{\Delta}_{\tau+1}+\cdots+\boldsymbol{\Delta}_{t-2})^\intercal\boldsymbol{H}_\tau^\tau\boldsymbol{\Delta}_{t-1} \neq \mathbf{0}^\intercal\boldsymbol{\Delta}_{t-1}$$

We compare this result with the value of $\mathcal{E}(t-1)$ given by Equation 3:

$$\mathcal{E}(t-1) = \frac{1}{t-1}\Bigg((t-2)\cdot\mathcal{E}(t-2) + \frac{1}{2}\boldsymbol{\Delta}_{t-1}^\intercal(\sum_{o=1}^{t-2}\mathbf{H}_o^\star)\boldsymbol{\Delta}_{t-1} + \sum_{\tau=1}^{t-2}(\boldsymbol{\Delta}_{\tau+1}+\cdots+\boldsymbol{\Delta}_{t-1})^\intercal\boldsymbol{H}_\tau^\star\,\boldsymbol{\Delta}_{t-1}\Bigg)$$
$$= \frac{1}{t-1}\Bigg(0 + \frac{1}{2}\boldsymbol{\Delta}_{t-1}^\intercal(\sum_{\tau=1}^{t-2}\boldsymbol{H}_\tau^\star)\boldsymbol{\Delta}_{t-1} + \sum_{\tau=1}^{t-2}(\boldsymbol{\Delta}_{\tau+1}+\cdots+\boldsymbol{\Delta}_{t-1})^\intercal\boldsymbol{H}_\tau^\star\,\boldsymbol{\Delta}_{t-1}\Bigg) \neq 0$$

We arrived at a contradiction, which proves the induction step and concludes the claim's proof.

A.1.3 THEOREM 2

In order to arrive at the statement of Theorem 2 simply notice that, by definition, $\mathcal{E}(1) = 0$ for any continual learning algorithm, and apply repeatedly Theorem 1. We briefly go through all the steps.

By $\mathcal{E}(1) = 0$ and Theorem 1, $\mathcal{E}(2) = \boldsymbol{\Delta}_2^\intercal(\sum_{o=1}^{1}\mathbf{H}_o^\star)\boldsymbol{\Delta}_2$. By assumption 1-2, all Hessian matrices are positive semi-definite and, accordingly, $\mathcal{E}(2) \geq 0$. It follows that $\mathcal{A}$ is forgetting-optimal if and only if $\mathcal{E}(2) = 0$. Recursively applying this argument, if $\mathcal{A}$ is forgetting-optimal then it holds:

$$\mathcal{E}(2) = \boldsymbol{\Delta}_2^\intercal(\sum_{o=1}^{1}\mathbf{H}_o^\star)\boldsymbol{\Delta}_2 = 0$$
$$\cdots$$
$$\mathcal{E}(T) = \boldsymbol{\Delta}_T^\intercal(\sum_{o=1}^{T-1}\mathbf{H}_o^\star)\boldsymbol{\Delta}_T = 0$$

as stated in Theorem 2.

A.1.4 THEOREM 3

We wish to prove the following statement regarding Orthogonal Gradient Descent (OGD) Farajtabar et al. (2020).

Theorem 3. Let $\mathcal{A} : [T] \to \Theta$ be a continual learning algorithm satisfying Assumption 1. If $\mathcal{A}(t) = \boldsymbol{\Delta}_t$ satisfies Equation 5, then it satisfies Equation 4.

We start by recalling the OGD algorithm. Let $D_o = \{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_{n_o}, y_{n_o}))\}$ be the dataset associated with task $\mathcal{T}_o$ and $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) \in \mathbb{R}^K$ be the network output corresponding to the input $\boldsymbol{x}$. The gradient of the network output with respect to the network parameters is the $P \times K$ matrix:

$$\nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \left[\nabla_{\boldsymbol{\theta}} \boldsymbol{f}^1_{\boldsymbol{\theta}}(\boldsymbol{x}), \cdots, \nabla_{\boldsymbol{\theta}} \boldsymbol{f}^K_{\boldsymbol{\theta}}(\boldsymbol{x})\right]$$

The standard version of OGD imposes the following constraint on any parameter update $\boldsymbol{u}$:

$$\langle \boldsymbol{u}, \nabla_{\boldsymbol{\theta}} \boldsymbol{f}^k_{\boldsymbol{\theta}_o}(\boldsymbol{x}_i)\rangle = 0$$

for all $k \in [1, K]$, $\boldsymbol{x}_i \in D_o$ and $o \leq t$, t being the number of tasks solved so far. If SGD is used, for example, the update $\boldsymbol{u}$ is the gradient of the new task loss for a data batch. Notice that the old task gradients are evaluated at the minima $\boldsymbol{\theta}_o$.

Proof Recall the definition of the average loss:

$$\mathcal{L}_t(\boldsymbol{\theta}) = \frac{1}{n_t} \sum_{i=0}^{n_t} l_t(\boldsymbol{x}_{i,t}, y_{i,t}, \boldsymbol{\theta})$$

The Hessian matrix of the loss $\boldsymbol{H}_t(\boldsymbol{\theta})$ can be decomposed as a sum of two other matrices (Schraudolph, 2002): the *outer-product* Hessian and the *functional* Hessian.

$$\boldsymbol{H}_t(\boldsymbol{\theta}) = \frac{1}{n_t} \sum_{i=1}^{n_t} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \left[\nabla^2_{\boldsymbol{f}} \ell_i\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)^\intercal + \frac{1}{n_t} \sum_{i=1}^{n_t} \sum_{k=1}^{K} \left[\nabla_{\boldsymbol{f}} \ell_i\right]_k \nabla^2_{\boldsymbol{\theta}} \boldsymbol{f}^k_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \,, \tag{9}$$

where $\ell_i = l(\boldsymbol{x_i}, y_i)$ and $\nabla^2_{\boldsymbol{f}} \ell_i$ is the $K \times K$ matrix of second order derivatives of the loss ℓ_i with respect to the network output $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x_i})$. At the optimum $\boldsymbol{\theta}_t$ (Assumption 1) the contribution of the functional Hessian is negligible (Singh et al., 2021). We rewrite the goal of the proof using these two facts:

$$\boldsymbol{\Delta}_t^\intercal \mathbf{H}^\star_o \boldsymbol{\Delta}_t = 0$$

$$\boldsymbol{\Delta}_t^\intercal \left(\frac{1}{n_o} \sum_{i=1}^{n_o} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \left[\nabla^2_{\boldsymbol{f}} \ell_i\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)^\intercal \right) \boldsymbol{\Delta}_t = 0 \tag{10}$$

Farajtabar et al. (2020) apply the OGD constraint (Equation 5) to the batch gradient vector $\boldsymbol{g}_B = \nabla \mathcal{L}^B_t$. Following this choice $\boldsymbol{\Delta}_t = \sum_{s=1}^{S_t} -\eta \boldsymbol{g}_s$, where η is the learning rate. Clearly, if, for all s, $\boldsymbol{g}_s$ satisfies Equation 5, then $\boldsymbol{\Delta}_t$ satisfies it. Hereafter, we ignore the specific form of $\boldsymbol{\Delta}_t$, proving the result for a broader class of algorithms for which $\boldsymbol{\Delta}_t$ satisfies the OGD constraint. Continuing from Equation 10:

$$\frac{1}{n_o} \left(\sum_{i=1}^{n_o} \nabla_{\boldsymbol{\theta}} \boldsymbol{\Delta}_t^\intercal \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \left[\nabla^2_{\boldsymbol{f}} \ell_i\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)^\intercal \boldsymbol{\Delta}_t \right)$$

$$\frac{1}{n_o} \left(\sum_{i=1}^{n_o} \nabla_{\boldsymbol{\theta}} \left[\boldsymbol{\Delta}_t^\intercal \left[\nabla_{\boldsymbol{\theta}} \boldsymbol{f}^1_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \cdots, \nabla_{\boldsymbol{\theta}} \boldsymbol{f}^K_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right]\right] \left[\nabla^2_{\boldsymbol{f}} \ell_i\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)^\intercal \boldsymbol{\Delta}_t \right)$$

$$\frac{1}{n_o} \left(\sum_{i=1}^{n_o} \nabla_{\boldsymbol{\theta}} \left[\underbrace{\boldsymbol{\Delta}_t^\intercal \nabla_{\boldsymbol{\theta}} \boldsymbol{f}^1_{\boldsymbol{\theta}}(\boldsymbol{x}_i)}_{=0}, \cdots, \underbrace{\boldsymbol{\Delta}_t^\intercal \nabla_{\boldsymbol{\theta}} \boldsymbol{f}^K_{\boldsymbol{\theta}}(\boldsymbol{x}_i)}_{=0} \right] \left[\nabla^2_{\boldsymbol{f}} \ell_i\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_i)^\intercal \boldsymbol{\Delta}_t \right) = 0,$$

where $\boldsymbol{\Delta}_t^\intercal \nabla_{\boldsymbol{\theta}} \boldsymbol{f}^k_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = 0$ is the OGD constraint, which holds for any k, i and $o < t$. This concludes the proof.

A.1.5 COROLLARY 1

Consider an algorithm $\mathcal{A}$ satisfying Assumptions 1-2. By Theorem 3 we know that, if $\mathcal{A}(t) = \boldsymbol{\Delta}_t$ satisfies the OGD constraint, then $\boldsymbol{\Delta}_t^\intercal \mathbf{H}^\star_o \boldsymbol{\Delta}_t = 0 \, \forall o < t$. Consequently, $\boldsymbol{\Delta}_t^\intercal (\sum_{o=1}^{t-1} \mathbf{H}^\star_o) \boldsymbol{\Delta}_t = 0$. By Theorem 2, it follows that $\mathcal{E}(t) = 0 \; \forall \, t \in [T]$.

A.1.6 ON THE VALIDITY OF OGD-GTL

Instead of considering all the function gradients with respect to all the outputs $\{\nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}^k(\boldsymbol{x}) \,|\, k \in [1, K], \boldsymbol{x} \in D_o\}$, Farajtabar et al. (2020) also consider a cheaper approximation where they impose orthogonality only with respect to the index corresponding to the true ground truth label (GTL). We show this can be understood via fairly mild assumptions, if the loss function is cross-entropy.

For a cross-entropy loss, $\boldsymbol{f}_{\boldsymbol{\theta}}^k(\boldsymbol{x_i})$ is the log-probability (or logit) associated with class k for input $\boldsymbol{x_i}$. The probability $p(y_i = k|\boldsymbol{x_i}; \boldsymbol{\theta})$ is then defined as $(\boldsymbol{p_i})_j = \mathrm{softmax}(\boldsymbol{f_\theta}(\boldsymbol{x_i}))_k$. Recall, from the proof of Theorem 3, that the theorem statement can be equivalently written as:

$$\boldsymbol{\Delta}_t^\intercal \left(\frac{1}{n_o} \sum_{i=1}^{n_o} \nabla_{\boldsymbol{\theta}} \boldsymbol{f_\theta}(\boldsymbol{x}_i) \left[\nabla^2_{\boldsymbol{f}} \ell_i \right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f_\theta}\left(\boldsymbol{x}_i\right)^\intercal \right) \boldsymbol{\Delta}_t = 0$$

For a cross-entropy loss the Hessian of the loss with respect to the network output is given by:

$$\nabla^2_{\boldsymbol{f}} \ell_i = \mathrm{diag}(\boldsymbol{p_i}) - \boldsymbol{p_i} \boldsymbol{p_i}^\intercal$$

Without loss of generality, assume index $k = 1$ corresponds to the GTL. Towards the end of the usual training, the softmax output at index 1, i.e., $p_1 \approx 1$. Let us assume that the probabilities for the remaining output coordinates is equally split between them. More precisely, let

$$p_2, \cdots, p_K = \frac{1 - p_1}{K - 1}.$$

Hence, in terms of their scales, we can regard $p_1 = \mathcal{O}(1)$ while $p_2, \cdots, p_K = \mathcal{O}\left(K^{-1}\right)$. Furthermore, $(1 - p_i) = \mathcal{O}(1) \quad \forall\, i \in [2 \cdots K]$. Then, a simple computation of the inner matrix in the Hessian outer product, i.e., $\nabla^2_{\boldsymbol{f}} \ell = \mathrm{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top$, will reveal that *diagonal entry corresponding to the ground truth label is* $\mathcal{O}(1)$*, while rest of the entries in the matrix are of the order* $\mathcal{O}\left(K^{-1}\right)$ *or* $\mathcal{O}\left(K^{-2}\right)$ — thereby explaining this approximation. Also, we can see that this approximation would work well only when $K \gg 1$ and does not carry over to other loss functions, e.g. Mean Squared Error (MSE). In fact, $K = 2$ all the entries of this matrix are of the same magnitude $|p_1(1 - p_1)|$, and for MSE simply $\nabla^2_{\boldsymbol{f}} \ell_i = \boldsymbol{I}_K$.

A.1.7 THEOREM 5

We start by proving Theorem 5. Before proceeding, we introduce the special notation to be used in the proof.

Notation: given some $a \times b$ matrix $\boldsymbol{C}$ we identify with $\mathrm{vec}(\boldsymbol{C})$ its $a \cdot b$ vector form ($\mathrm{vec}(\cdot)$ is the *vectorisation* operator). We use the standard *input*$\times$*output* in order to determine the matrices dimensionality. Moreover, for simplicity, we consider all the network layers to have width D (the same analysis can be done for layers of different widths). As an example, the gradient $\nabla_{\boldsymbol{W}^l} \mathcal{L}_t(\boldsymbol{\theta})$ is a tensor of dimensions $D \times D \times 1$, and $\mathrm{vec}(\nabla_{\boldsymbol{W}^l} \mathcal{L}_t(\boldsymbol{\theta}))$ is a vector of size D^2. Finally, we use $\boldsymbol{\theta}$ to refer to the *vector* of network parameters (the vectorization operation is implicit), thus $\nabla_{\boldsymbol{\theta}} \mathcal{L}_t(\boldsymbol{\theta})$, shortened to $\nabla \mathcal{L}_t(\boldsymbol{\theta})$ is a vector of size P.

Proof We recall the GPM algorithm. Let $\boldsymbol{x}_i^{l+1} = \sigma(\boldsymbol{W}^{l\intercal} \boldsymbol{x}^l)$ be the representation of $\boldsymbol{x}_i$, by layer l of the network ($\boldsymbol{x}_i^0 = \boldsymbol{x}_i$). Notice that the GPM algorithm does not allow for bias vectors in the layer functions, thus the set of network parameters is $\{\boldsymbol{W}^0, \ldots, \boldsymbol{W}^{L-1}\}$. The GPM constraint on the weight matrix update $\boldsymbol{\Delta W}^l$ reads:

$$\langle \boldsymbol{\Delta W}^l, \boldsymbol{x}_i^l \rangle = 0 \tag{11}$$

for all network layers l, $\boldsymbol{x}_i \in D_o$ and all previous tasks $\mathcal{T}_o$, $o < t$.

The optimization path from $\boldsymbol{\theta}_{t-1}$ to $\boldsymbol{\theta}_t$ is the sequence of parameter values $\{\boldsymbol{\theta}^1_{t-1\to t}, \ldots, \boldsymbol{\theta}^{S_t}_{t-1\to t}\}$, with parameter updates $\boldsymbol{\delta}_t^{(i)} = \boldsymbol{\theta}^i_{t-1\to t} - \boldsymbol{\theta}^{i-1}_{t-1\to t}$. In the following discussion we drop the task index and simply write $\boldsymbol{\delta}^{(i)} = \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^{(i-1)}$. Similarly, $\boldsymbol{W}^{l(i)} = \boldsymbol{W}^{l(i-1)} + \boldsymbol{\delta}_t^{(i)}[l]$ for the layer parameters. Observe that:

$$\mathrm{vec}(\nabla_{\boldsymbol{W}^l} \mathcal{L}_o(\boldsymbol{\theta}^{(i-1)}))^\intercal \,\mathrm{vec}(\boldsymbol{\delta}^{(i)}[l]) = 0 \;\forall\, l \implies \nabla \mathcal{L}_o(\boldsymbol{\theta}^{(i-1)})^\intercal \boldsymbol{\delta}^{(i)} = 0$$

and, similarly:

$$\text{vec}(\boldsymbol{\delta}_t^{(i)}[l])^\intercal \mathbf{H}_o(\boldsymbol{W}^{l(i-1)})\,\text{vec}(\boldsymbol{\delta}_t^{(i)}[l]) = 0\ \forall\, l$$
$$\implies \boldsymbol{\delta}_t^{(i)\intercal}\,\text{block-diag}(\mathbf{H}_o(\boldsymbol{\theta}_{t-1\to t}^{(i-1)}))\,\boldsymbol{\delta}_t^{(i)} = 0$$

Thus it is sufficient to prove the following two statements, for all $i \in [1, S_t]$, all layers l, and all $o < t$:

$$\text{vec}(\nabla_{\boldsymbol{W}^l}\mathcal{L}_o(\boldsymbol{\theta}^{(i-1)}))^\intercal\,\text{vec}(\boldsymbol{\delta}^{(i)}[l]) = 0 \tag{12}$$
$$\text{vec}(\boldsymbol{\delta}_t^{(i)}[l])^\intercal \mathbf{H}_o(\boldsymbol{W}^{l(i-1)})\,\text{vec}(\boldsymbol{\delta}_t^{(i)}[l]) = 0 \tag{13}$$

By the linearity of the derivative $\nabla_{\boldsymbol{W}^l}\mathcal{L}_o(\boldsymbol{\theta}^{(i-1)}) = \frac{1}{n_o}\sum_{j=1}^{n_o}\nabla_{\boldsymbol{W}^l}\ell_j$, where $\ell_j = l_o(\boldsymbol{x_j}, y_j; \boldsymbol{\theta}^{(i-1)})$. We unpack $\nabla_{\boldsymbol{W}^l}\ell_j$ using the chain rule:

$$\nabla_{\boldsymbol{W}^l}\ell_j = \nabla_{\boldsymbol{W}^l}\boldsymbol{f}_{\boldsymbol{\theta}^{(i-1)}}(\boldsymbol{x_j}) \circ \nabla_{\boldsymbol{f}}\ell_j$$

Again, by the use of the chain rule we have:

$$\nabla_{\text{vec}(\boldsymbol{W}^l)}\boldsymbol{f}_{\boldsymbol{\theta}^{(i-1)}}(\boldsymbol{x_j}) = \nabla_{\text{vec}(\boldsymbol{W}^l)}\left(\boldsymbol{W}^{l^{(i-1)\intercal}}\boldsymbol{x}_j^{l(i-1)}\right)\underbrace{\text{diag}\left(\sigma'(\boldsymbol{W}^{l^{(i-1)\intercal}}\boldsymbol{x}_j^{l(i-1)})\right)}_{=:\boldsymbol{A}}\cdot\underbrace{\nabla_{\boldsymbol{x}_j^{l+1}}\boldsymbol{f}_{\boldsymbol{\theta}^{(i-1)}}(\boldsymbol{x}_j)}_{=:\boldsymbol{\zeta}_{l+1}}$$
$$= \nabla_{\text{vec}(\boldsymbol{W}^l)}\left(\boldsymbol{W}^{l^{(i-1)\intercal}}\boldsymbol{x}_j^{l(i-1)}\right)\cdot\boldsymbol{A}\cdot\boldsymbol{\zeta}_{l+1}$$
$$= (\boldsymbol{x}_j^{l(i-1)}\otimes\boldsymbol{I}_D)\cdot\boldsymbol{A}\cdot\boldsymbol{\zeta}_{l+1}$$
$$= (\boldsymbol{x}_j^{l(i-1)}\otimes\boldsymbol{A})\cdot\boldsymbol{\zeta}_{l+1}$$

where $\boldsymbol{A}$ contains the element-wise derivatives of the activation function σ and $\boldsymbol{I}_D$ is the $D \times D$ identity matrix. Expanding recursively the layer activation:

$$\boldsymbol{x}_j^{l(i-1)} = \sigma(\boldsymbol{W}^{l-1(i-1)^\intercal}\boldsymbol{x}_j^{l-1(i-1)})$$
$$\dots$$
$$\boldsymbol{x}_j^{1^{(i-1)}} = \sigma(\boldsymbol{W}^{0^{(i-1)\intercal}}\boldsymbol{x}_j)$$

As a consequence of the GPM constraint:

$$\boldsymbol{W}^{0^{(i-1)\intercal}}\boldsymbol{x}_j = \left(\boldsymbol{W}_o^0 + (\boldsymbol{W}^{0^{(i-1)}} - \boldsymbol{W}_o^0)\right)^\intercal\boldsymbol{x}_j = \boldsymbol{W}_o^{0\intercal}\boldsymbol{x}_j,$$

$\boldsymbol{W}_o^0$ being the value of $\boldsymbol{W}^0$ at the optimum of task $\mathcal{T}_o$, i.e. $\boldsymbol{\theta}_o$. By a recursive argument we then get:

$$\boldsymbol{x}_j^{1^{(i-1)}} = \sigma(\boldsymbol{W}_o^{0\intercal}\boldsymbol{x}_j) = \boldsymbol{x}_j^1$$
$$\dots$$
$$\boldsymbol{x}_j^{l(i-1)} = \sigma(\boldsymbol{W}_o^{l-1\intercal}\boldsymbol{x}_j^{l-1}) = \boldsymbol{x}_j^l$$

In a nutshell, the GPM constraint freezes the network activation vectors to their value at the optimum, for each input stored in memory. Using this result, the expression of the output gradient simplifies to:

$$\nabla_{\text{vec}(\boldsymbol{W}^l)}\boldsymbol{f}_{\boldsymbol{\theta}^{(i-1)}}(\boldsymbol{x_j}) = (\boldsymbol{x}_j^l\otimes\boldsymbol{A})\cdot\boldsymbol{\zeta}_{l+1} \tag{14}$$

We are now ready finalize the first point of our proof (Equation 12).

$$\text{vec}(\nabla_{\boldsymbol{W}^l}\mathcal{L}_o(\boldsymbol{\theta}^{(i-1)}))^\intercal\,\text{vec}(\boldsymbol{\delta}^{(i)}[l]) =$$
$$(\nabla_{\text{vec}(\boldsymbol{W}^l)}\mathcal{L}_o(\boldsymbol{\theta}^{(i-1)}))^\intercal\,\text{vec}(\boldsymbol{\delta}^{(i)}[l]) =$$
$$\left(\frac{1}{n_o}\sum_{j=1}^{n_o}(\boldsymbol{x}_j^l\otimes\boldsymbol{A})\cdot\boldsymbol{\zeta}_{l+1}\cdot\nabla_{\boldsymbol{f}}\ell_j\right)^\intercal\text{vec}(\boldsymbol{\delta}^{(i)}[l]) =$$
$$\frac{1}{n_o}\sum_{j=1}^{n_o}\text{vec}(\boldsymbol{\delta}^{(i)}[l])^\intercal(\boldsymbol{x}_j^l\otimes\boldsymbol{A})\cdot\boldsymbol{\zeta}_{l+1}\cdot\nabla_{\boldsymbol{f}}\ell_j =$$
$$\frac{1}{n_o}\sum_{j=1}^{n_o}\text{vec}(\underbrace{\boldsymbol{x}_j^{l\intercal}\boldsymbol{\delta}^{(i)}[l]}_{=\mathbf{0}}\boldsymbol{A})\cdot\boldsymbol{\zeta}_{l+1}\cdot\nabla_{\boldsymbol{f}}\ell_j = 0 \qquad \text{(GPM constraint, eq. 11)}$$

We now proceed to the second point. As before, we decompose the Hessian into the sum of outer-product and functional Hessian:

$$\boldsymbol{H}_o(\boldsymbol{\theta}) = \frac{1}{n_o}\sum_{j=1}^{n_o} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_j) \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_j)^\intercal + \frac{1}{n_o}\sum_{i=j}^{n_o}\sum_{k=1}^{K} \left[\nabla_{\boldsymbol{f}} \ell_j\right]_k \nabla^2_{\boldsymbol{\theta}} \boldsymbol{f}^k_{\boldsymbol{\theta}}(\boldsymbol{x}_j)$$

In particular we are interested in the block-diagonal elements of the Hessian matrix, i.e. $\mathbf{H}_o(\boldsymbol{W}^{l(i-1)})$. When *ReLU* or linear activation functions are used the functional part does not contribute to the the block-diagonal elements, as it can be seen from Equation 14:

$$\begin{aligned}\nabla^2_{\boldsymbol{W}^l} \boldsymbol{f}^k_{\boldsymbol{\theta}^{(i-1)}}(\boldsymbol{x}_j) &= \nabla_{\boldsymbol{W}^l}\left((\boldsymbol{x}_j^l \otimes \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1}\right) \\ &= (\boldsymbol{x}_j^l \otimes [\nabla_{\boldsymbol{W}^l} \boldsymbol{A}]) \cdot \boldsymbol{\zeta}_{l+1}\end{aligned}$$

$\boldsymbol{A}$ is a diagonal matrix, and if $\sigma = \mathrm{ReLU}$ then:

$$\boldsymbol{A}_{kk} = \begin{cases} 1 & \text{if } [\boldsymbol{W}^{l(i-1)^\intercal} \boldsymbol{x}_j^{l(i-1)}]_k > 0 \\ 0 & \text{otherwise} \end{cases}$$

Consequently, the gradient with respect to the layer weights $\boldsymbol{W}^{l(i-1)}$ is null. Similarly, if the activation function is linear, $\boldsymbol{A}$ is simply the identity matrix, and the gradient is 0. For a general activation function σ, the diagonal entries of $\boldsymbol{A}$ can have a non-linear dependence on $\boldsymbol{W}^{l(i-1)}$, and a case-by-case evaluation is required. Hereafter, we consider the piecewise-linear and linear case, for which the functional Hessian is *block-hollow*.

We can thus ignore the outer-product component in the Hessian layer blocks $\mathbf{H}_o(\boldsymbol{W}^{l(i-1)})$ and write:

$$\begin{aligned}\mathbf{H}_o(\boldsymbol{W}^{l(i-1)}) &= \frac{1}{n_o}\sum_{j=1}^{n_o} \nabla_{\boldsymbol{W}^l} \boldsymbol{f}_{\boldsymbol{\theta}^{(i-1)}}(\boldsymbol{x}_j) \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \nabla_{\boldsymbol{W}^l} \boldsymbol{f}_{\boldsymbol{\theta}^{(i-1)}}(\boldsymbol{x}_j)^\intercal \\ &= \frac{1}{n_o}\sum_{j=1}^{n_o} \left[(\boldsymbol{x}_j^l \otimes \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1}\right] \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \left[(\boldsymbol{x}_j^l \otimes \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1}\right]^\intercal\end{aligned}$$

We can now finalise the second point in our proof (Equation 13).

$$\begin{aligned}&\mathrm{vec}(\boldsymbol{\delta}_t^{(i)}[l])^\intercal \mathbf{H}_o(\mathrm{vec}(\boldsymbol{W}^{l(i-1)}))\,\mathrm{vec}(\boldsymbol{\delta}_t^{(i)}[l]) = \\ &= \mathrm{vec}(\boldsymbol{\delta}_t^{(i)}[l])^\intercal \left(\frac{1}{n_o}\sum_{j=1}^{n_o} \left[(\boldsymbol{x}_j^l \otimes \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1}\right] \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \left[(\boldsymbol{x}_j^l \otimes \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1}\right]^\intercal\right) \mathrm{vec}(\boldsymbol{\delta}_t^{(i)}[l]) = \\ &= \frac{1}{n_o}\sum_{j=1}^{n_o} \left(\left[\mathrm{vec}(\boldsymbol{\delta}_t^{(i)}[l])^\intercal (\boldsymbol{x}_j^l \otimes \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1}\right]\right) \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \left(\left[(\boldsymbol{x}_j^l \otimes \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1}\right]^\intercal \mathrm{vec}(\boldsymbol{\delta}_t^{(i)}[l])\right) = \\ &= \frac{1}{n_o}\sum_{j=1}^{n_o} \left(\left[\mathrm{vec}(\underbrace{\boldsymbol{x}_j^{l^\intercal} \boldsymbol{\delta}_t^{(i)}[l]}_{=\mathbf{0}} \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1}\right]\right) \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \left(\left[(\boldsymbol{x}_j^l \otimes \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1}\right]^\intercal \mathrm{vec}(\boldsymbol{\delta}_t^{(i)}[l])\right) = 0\end{aligned}$$

Once more, $\boldsymbol{x}_j^{l^\intercal} \boldsymbol{\delta}_t^{(i)}[l]$ evaluates to 0 following the GPM constraint. We have evaluated only the right side of the expression (as it is enough to prove the equivalence), however notice that the expression is symmetric.

Notice that our proof applies to any $o < t$ and any $i \in [1, S_t]$ and layer l in the network, since the GPM constraint also holds for all these cases. We have proved Theorem 5 for all network with piecewise-linear and linear activation functions. We leave the case of other activation functions to the reader.

A.1.8 THEOREM 4

We are now ready to prove that, under Assumptions 1-2, algorithms satisfying the GPM constraint also satisfy the null-forgetting constraint (Equation 4, when the Hessian matrices are block diagonal.

Recall the null-forgetting constraint (Equation 4):

$$\mathcal{A}(t) = \boldsymbol{\Delta}_t = \operatorname{argmin} \mathcal{E}(t) \iff \boldsymbol{\Delta}_t^\intercal (\sum_{o=1}^{t-1} \mathbf{H}_o^\star) \boldsymbol{\Delta}_t = 0$$

By Assumption 1, we know $\mathbf{H}_o^\star$ coincides with the outer-product Hessian:

$$\boldsymbol{H}_o(\boldsymbol{\theta}) = \frac{1}{n_o} \sum_{j=1}^{n_o} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j) \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j)^\intercal$$

Thus the statement to prove is:

$$\begin{aligned}
&\boldsymbol{\Delta}_t^\intercal (\sum_{o=1}^{t-1} \mathbf{H}_o^\star) \boldsymbol{\Delta}_t = 0 \\
&\iff \boldsymbol{\Delta}_t^\intercal (\sum_{o=1}^{t-1} \frac{1}{n_o} \sum_{j=1}^{n_o} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j) \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j)^\intercal) \boldsymbol{\Delta}_t = 0 \\
&\iff \sum_{o=1}^{t-1} \frac{1}{n_o} \sum_{j=1}^{n_o} \boldsymbol{\Delta}_t^\intercal \left(\nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j) \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j)^\intercal\right) \boldsymbol{\Delta}_t = 0 \\
&\iff \sum_{o=1}^{t-1} \frac{1}{n_o} \sum_{j=1}^{n_o} (\sum_{i=1}^{S_t} \boldsymbol{\delta}_t^{(i)})^\intercal \left(\nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j) \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j)^\intercal\right) (\sum_{i=1}^{S_t} \boldsymbol{\delta}_t^{(i)}) = 0 \\
&\iff \sum_{o=1}^{t-1} \frac{1}{n_o} \sum_{j=1}^{n_o} \left(\sum_{i=1}^{S_t} \boldsymbol{\delta}_t^{(i)\intercal} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j)\right) \left[\nabla^2_{\boldsymbol{f}} \ell_j\right] \left(\sum_{i=1}^{S_t} \nabla_{\boldsymbol{\theta}} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j)^\intercal \boldsymbol{\delta}_t^{(i)}\right) = 0
\end{aligned}$$

Being the Hessian matrix block-diagonal by assumption, it is sufficient to prove:

$$\operatorname{vec}(\boldsymbol{\delta}_t^{(i)}[l])^\intercal \nabla_{\boldsymbol{W}^l} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j) = 0$$

where we adopt the notation from the proof of Theorem 5. Using the result in Equation 14:

$$\begin{aligned}
\operatorname{vec}(\boldsymbol{\delta}_t^{(i)}[l])^\intercal \nabla_{\boldsymbol{W}^l} \boldsymbol{f}_{\boldsymbol{\theta}_o}(\boldsymbol{x}_j) &= \operatorname{vec}(\boldsymbol{\delta}_t^{(i)}[l])^\intercal (\boldsymbol{x}_j^l \otimes \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1} \\
&= \operatorname{vec}(\boldsymbol{x}_j^{l\intercal} \boldsymbol{\delta}_t^{(i)}[l] \boldsymbol{A}) \cdot \boldsymbol{\zeta}_{l+1} = 0,
\end{aligned}$$

where the last equality follows from the GPM constraint.

A.1.9 Corollary 2

Finally we go over the proof of Corollary 2. We start by considering the following Taylor expansion:

$$\mathcal{E}_o(\boldsymbol{\theta}_{t-1\to t}^{(i)}) = \mathcal{E}_o(\boldsymbol{\theta}_{t-1\to t}^{(i-1)}) + \nabla \mathcal{L}_o(\boldsymbol{\theta}_{t-1\to t}^{(i-1)})^\intercal \boldsymbol{\delta}_t^{(i)} + \frac{1}{2} \boldsymbol{\delta}_t^{(i)\intercal} \mathbf{H}_o(\boldsymbol{\theta}_{t-1\to t}^{(i-1)}) \boldsymbol{\delta}_t^{(i)} + \mathcal{O}(\|\delta_t^{(i)}\|)$$

By the GPM constraint, $\nabla \mathcal{L}_o(\boldsymbol{\theta}_{t-1\to t}^{(i-1)})^\intercal \boldsymbol{\delta}_t^{(i)} = 0$ and $\boldsymbol{\delta}_t^{(i)\intercal} \mathbf{H}_o(\boldsymbol{\theta}_{t-1\to t}^{(i-1)}) \boldsymbol{\delta}_t^{(i)} = 0$, as by assumption $\operatorname{block-diag}(\mathbf{H}_o(\boldsymbol{\theta}_{t-1\to t}^{(i-1)}) = \mathbf{H}_o(\boldsymbol{\theta}_{t-1\to t}^{(i-1)})$. Consequently, the expression of forgetting on the optimization path simplifies to:

$$\mathcal{E}_o(\boldsymbol{\theta}_{t-1\to t}^{(i)}) = \mathcal{E}_o(\boldsymbol{\theta}_{t-1\to t}^{(i-1)}) + \mathcal{O}(\|\delta_t^{(i)}\|)$$

For sufficiently small step sizes (often $\|\delta_t^{(i)}\| \in \mathcal{O}(\eta)$, where η is the learning rate), the approximation error is negligible, and, by recursively applying the above identity we obtain: $\mathcal{E}_o(\boldsymbol{\theta}_t) = \mathcal{E}_o(\boldsymbol{\theta}_o) = 0$, which proves the statement.

A.2 Further discussion

A.2.1 Non monotonic null-forgetting

The null-forgetting constraint presupposes that forgetting is zero after learning each task, i.e. $\mathcal{E}_\tau(t) = 0$ for any τ and t (with $\tau < t$). Here we consider what happens when $\mathcal{E}(\tau) > 0$ for some $\tau < t$. Specifically, we want to derive a new constraint on the update $\boldsymbol{\Delta}_{\tau+1}$ such that $\mathcal{E}(\tau + 1) = 0$.

Theorem 6 (Non monotonic null-forgetting constraint.)**.** *Let $\mathcal{A} : [T] \to \Theta$ be a continual learning algorithm satisfying Assumptions 1-2. Furthermore, for the first $\tau - 1$ tasks it holds that $\mathcal{E}(k) = 0$. Let $\mathcal{A}(\tau) = \boldsymbol{\Delta}_\tau$ s.t. $\mathcal{E}(\tau) > 0$.*

Then for an update $\mathcal{A}(\tau + 1) = \boldsymbol{\Delta}_{\tau+1}$, $\mathcal{E}(\tau + 1) = 0$ if and only if:

$$\boldsymbol{\Delta}_{\tau+1} = \boldsymbol{z} - (\sum_{o=1}^{\tau} \mathbf{H}_o^\star)^\dagger \boldsymbol{B}\boldsymbol{\Delta}_\tau, \tag{16}$$

for any vector $\boldsymbol{z}$ satisfying the condition $(\sum_{o=1}^{\tau} \mathbf{H}_o^\star)\boldsymbol{z} = \mathbf{0}$. $\boldsymbol{B}$ is the matrix characterising $\mathcal{E}(\tau)$, i.e. $\mathcal{E}(\tau) = \frac{1}{2\tau}\boldsymbol{\Delta}_\tau^\intercal \boldsymbol{B}\boldsymbol{\Delta}_\tau$.

Proof. By Assumptions 1-2 we can approximate $\mathcal{E}(\tau + 1)$ using Equation 3:

$$\mathcal{E}(\tau+1) = \frac{1}{\tau+1}\left(\tau \cdot \mathcal{E}(\tau) + \frac{1}{2}\boldsymbol{\Delta}_{\tau+1}^\intercal(\sum_{o=1}^{\tau}\mathbf{H}_o^\star)\boldsymbol{\Delta}_{\tau+1} + \sum_{o=1}^{\tau-1}(\boldsymbol{\theta}_\tau - \boldsymbol{\theta}_o)^\intercal \mathbf{H}_o^\star \boldsymbol{\Delta}_{\tau+1}\right)$$

Using $\mathcal{E}(k) = 0 \,\forall\, k < \tau$ and Theorem 1 we have $\mathcal{E}(\tau) = \boldsymbol{\Delta}_\tau^\intercal \boldsymbol{B}\boldsymbol{\Delta}_\tau = C > 0$, where we define $\boldsymbol{B} := (\sum_{o=1}^{\tau-1} \mathbf{H}_o^\star)$. Moreover, notice that:

$$\boldsymbol{v}_\tau := \sum_{o=1}^{\tau-1}(\boldsymbol{\theta}_\tau - \boldsymbol{\theta}_o)^\intercal \mathbf{H}_o = \sum_{o=1}^{\tau-1}\boldsymbol{\Delta}_\tau^\intercal \mathbf{H}_o^\star + \boldsymbol{v}_{\tau-1}$$

From Theorem 1 it follows that $\boldsymbol{v}_{\tau-1} = \mathbf{0}$ and thus:

$$\mathcal{E}(\tau+1) = \frac{1}{\tau+1}\left(\tau \cdot C + \frac{1}{2}\boldsymbol{\Delta}_{\tau+1}^\intercal(\sum_{o=1}^{\tau}\mathbf{H}_o^\star)\boldsymbol{\Delta}_{\tau+1} + \boldsymbol{\Delta}_\tau^\intercal \mathbf{B}\boldsymbol{\Delta}_{\tau+1}\right)$$

Notice that this expression is quadratic in $\boldsymbol{\Delta}_{\tau+1}$ and thus $\mathcal{E}(\tau + 1)$ admits a closed-form solution. We get:

$$\text{argmin}_{\boldsymbol{\Delta}_{\tau+1}} \mathcal{E}(\tau+1) = \boldsymbol{z} - (\sum_{o=1}^{\tau}\mathbf{H}_o^\star)^\dagger \boldsymbol{B}\boldsymbol{\Delta}_\tau,$$

where $\boldsymbol{z}$ is any vector satisfying the condition $(\sum_{o=1}^{\tau}\mathbf{H}_o^\star)\boldsymbol{z} = 0$. Notice that if $\mathcal{E}(\tau) = 0$ then $\boldsymbol{B}\Delta_\tau = \mathbf{0}$ and 6 reduces to the null-forgetting constraint from Theorem 2. Intuitively, in order to reverse forgetting, the algorithm has to retrace its steps along the harmful directions in the previous update (i.e. $\boldsymbol{B}\boldsymbol{\Delta}_\tau$).

The next Corollary extends the result in Theorem 1 to the case of non-monotonic forgetting. It follows directly from Theorem 6 and the formulation of forgetting in Equation 3.

Corollary 3 (Non monotonic null-forgetting.)**.** *For any continual learning algorithm satisfying Assumptions 1-2 the following relationship between previous and current forgetting exists:*

$$\mathcal{E}(1), \ldots, \mathcal{E}(t-2) = 0, \mathcal{E}(t-1) > 0, \mathcal{E}(t) = 0 \implies \mathcal{E}(t+2) = \frac{1}{2}\boldsymbol{\Delta}_{t+2}^\intercal\left(\frac{1}{t+2}\cdot\sum_{o=1}^{t+1}\mathbf{H}_o^\star\right)\boldsymbol{\Delta}_{t+2}$$

A.2.2 Implications of null-forgetting

Theorem 1 describes forgetting in terms of the alignment between the current parameter update and the column space of the Hessians of the previous tasks.

Let $\overline{\mathbf{H}}_{<t}^\star := \frac{1}{t-1}\cdot\sum_{o=1}^{t-1}\mathbf{H}_o^\star$ denote the average Hessian matrix of the old tasks. Intuitively, the higher the rank of $\overline{\mathbf{H}}_{<t}^\star$ the less capacity is left to learn a new task. In order to see this, simply consider the constrained optimization problem:

$$\boldsymbol{\Delta}_t = \text{argmin}_{\boldsymbol{\Delta}\in\Theta}\, \mathcal{L}_t(\boldsymbol{\Delta} + \boldsymbol{\theta}_{t-1})$$
$$s.t.\ \boldsymbol{\Delta}_t^\intercal \overline{\mathbf{H}}_{<t}^\star \boldsymbol{\Delta}_t = 0$$

The constraint on the parameter update effectively restricts the search space from Θ to the $P - R$-dimensional subspace orthogonal to the column space of $\overline{\mathbf{H}}_{<t}^\star$, where $R = \text{rank}(\overline{\mathbf{H}}_{<t}^\star)$. Thus, the

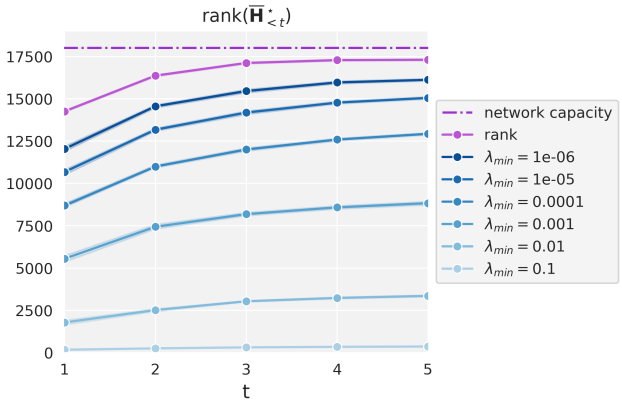

Figure 5: Rank and effective rank for various threshold λ values on the Rotated MNIST challenge, learned by SGD. The values are averaged over seeds.

higher R, the smaller the set of feasible solutions for the current task. Importantly, this formulation ignores potential *constructive interference* between tasks, which is a limitation of many parameter isolation models (Lin et al. (2022) do follow-up work on GPM addressing this problem). This condition is necessary under Assumption 1-2, where forgetting is always non-negative and thus there can be no constructive interference. However, in the more general setting, the condition reflects a worse-case scenario, and exploiting a constructive intereference of task may save network capacity for future tasks.

Methods enforcing sparsity on the parameter updates (Schwarz et al., 2021) induce a transformation of the loss gradient vector and Hessian matrix similar to parameter isolation methods. In practice, the column space of $\boldsymbol{H}_o$ is a subset of the task parameters Θ_o. Thereby, the alignment of the current update to the column space of the average Hessian becomes simply a function of the overlap with the previous updates $\|\boldsymbol{\Delta}_t - \boldsymbol{\theta}_{t-1}\|_0$ and the average Hessian rank is $R \leq (\sum_{o=1}^{t-1} r_o) \cdot P$, where r_o is the fraction of active connections for task $\mathcal{T}_o$. If the average Hessian rank is not maximal, Theorem 1 suggests that $r_t > (1 - (\sum_{o=1}^{t-1} r_o))$ can be achieved without suffering forgetting.

In Figure 5 we plot the evolution of $\mathrm{rank}(\overline{\mathbf{H}}^{\star}_{<t})$ over t for the Rotated-MNIST challenge. Additionally, we evaluate the effective rank of the average Hessian for multiple threshold λ values, which better captures the effective dimensionality of the matrix column space.

B Experimental Details

B.0.1 Experimental setup

The experiments are designed to support the theoretical contribution of the paper. The specific experimental design choices are motivated by the theory and by the common practices in the field. Especially when providing results on existing methods (Section 5.2), we employ similar experimental setting as in the original paper. Following is a detailed description of our experimental setup, which is aimed at facilitating reproducibility. To that aim, we also provide public access to our code[1].

Datasets.

1. *Rotated-MNIST*: a sequence of 5 tasks, each corresponding to a fixed rotation applied to the MNIST dataset. We fix the set of rotations to $[-45°, -22.5°, 0, 22.5°, 45°]$, in line with (Farajtabar et al., 2020). In order to be able to compute the network Hessian matrix, the input is downscaled to 14×14. The output space is shared across task, and has 10 dimensions, one for each digit. We employ the standard Pytorch train-test split, loading $60,000$ inputs to train each task.
2. *Split CIFAR-10*: a sequence of 5 tasks, each corresponding to a different pair of labels in the CIFAR-10 dataset. The splits are fixed across experiments: classes $(0, 1)$ for the first

[1]We hide the repository link for the double-blind review.

task, $(2, 3)$ for the second one, and so on. The output space is shared across task, and has dimensionality 2. We employ the standard Pytorch train-test split, loading $5,000 \times 2$ inputs to train each task.

3. *Split CIFAR-100*: a sequence of 20 tasks. Similarly to Split CIFARO-10, each task corresponds to a different set of 5 classes from the CIFAR-100 dataset. The splits are fixed across experiments. The output space is shared across task, and has dimensionality 5. We employ the standard Pytorch train-test split, loading 500×5 inputs to train each task.

Network architectures. Overall, we employ 3 different network architectures in the experiments: an MLP, a CNN, and a Transformer. We here describe the implementation of the three networks. In Table 3 we provide an overview of the different hyperparameter settings used: each experiment configuration takes one row of the table.

1. The MLP has 3 hidden layers with 50 units each. Experiments with similar MLP architectures are widespread in the literature (Farajtabar et al., 2020; Saha et al., 2021; Mirzadeh et al., 2020b; Hsu et al., 2018). However, we reduce the conventional layer width from 100 (or 200) to 50 neurons, while increasing the depth by one layer (from 2 to 3), thereby lowering the network size. Our network has a total of $18,010$ parameters. In the GPM experiments we do not include bias terms in the network, thus reducing the size further to $17,800$. We train this network using SGD, SGD†, OGD and GPM with the same hyperparameter configurations.
2. The CNN used is a ResNet 18 (He et al., 2016). We adopt the same implementation as (Mirzadeh et al., 2020b; Saha et al., 2021; Lopez-Paz & Ranzato, 2017), which is a smaller version of the original ResNet18, with three times less feature maps per layer. In total, the network has around 0.5M parameters. This network is trained using SGD and SGD† on Split CIFAR-10 and Split CIFAR-100 (the output layer size is changed accordingly). The same hyperparameter configuration is used for both SGD and SGD† in order to make a fair comparison.
3. The Transformer is a ViT (Dosovitskiy et al., 2020). The use of Transformers is not as widespread yet in the continual learning literature. We refer to a publicly available implementation of a ViT in Pytorch[2], which is again a smaller version than the original. The network has around 12M parameters. We retain the standard configuration provided in the reference implementation for almost all hyperparameters, except we simplify the optimization scheme. We use SGD instead of the Adam optimizer, no weight decay or label smoothing and a simple two-steps learning rate decay schedule. We train this network for SGD and SGD† again on Split CIFAR-10 and Split CIFAR-100, changing the output layer accordingly.

Metrics. Finally, we recapitulate the metrics used in the paper. We measure forgetting on a specific task as $\mathcal{E}_o(t) = \mathcal{L}_o(\boldsymbol{\theta}_t) - \mathcal{L}_o^\star$ and average forgetting as $\mathcal{E}(t) = \frac{1}{t}\sum_{o=1}^{t-1} \mathcal{E}_o(t)$. In the literature, forgetting is often measured as $\text{BWT} = \frac{1}{T-1}\sum_{o=1}^{T-1}(a_{o,o} - a_{T,o})$, where $a_{j,i}$ denotes the accuracy on task $\mathcal{T}_i$ for $\boldsymbol{\theta} = \boldsymbol{\theta}_j$. In our results we also report forgetting in terms of accuracy, in order to increase the interpretability of the scores. Simply, we replace the definition of forgetting with $\mathcal{E}_o(t) = a_{o,o} - a_{t,o}$ and $\mathcal{E}(t) = \frac{1}{t}\sum_{o=1}^{t-1} \mathcal{E}_o^a(t)$. Finally, in accordance with the convention in the literature, we also measure the average accuracy over tasks as $a(t) = \frac{1}{t}\sum_{o=1}^{t} a_{t,o}$. For all these scores we evaluate the loss and accuracy on the test set.

Selection of k in experiment 5.2 When comparing OGD, GPM and SGD† we have to be careful in selecting the number of new vectors to store in the memory for each task, since each method uses different vectors in practice. We decide to threshold the spectral energy of the addition to the memory, using ϵ as a cutoff parameter. In practice, instead of controlling , the number of eigenvectors, we control , the portion of spectral energy left out, where spectral energy is the squared eigenvalues norm . From the eigendecomposition, we obtain the cumulative spectral energy and we make a cut keeping

[2]Found on Github: https://github.com/omihub777/ViT-CIFAR. Last accessed in May 2023.

Table 3: Experiments configurations

Network	Dataset	Algorithm	Learning rate	Epochs
MLP	Rotated-MNIST	SGD	0.01	5
		SGD	$\{0.01, 10^{-5}\}$	15
		SGD†, $\epsilon = 0.01$	0.01	5
		SGD†, $\epsilon = 0.01$	$\{0.01, 10^{-5}\}$	15
		GPM, $\epsilon = 0.01$	0.01	5
		GPM, $\epsilon = 0.01$	$\{0.01, 10^{-5}\}$	15
		OGD-gtl, $\epsilon = 0.01$	0.01	5
		OGD-gtl, $\epsilon = 0.01$	$\{0.01, 10^{-5}\}$	15
ResNet18	Split CIFAR-10	SGD	$\{0.1, 0.001\}$	$\{10/20, 1\}$
		SGD†, $k = 10$	$\{0.1, 0.001\}$	$\{10/20, 1\}$
		SGD†, $k = 20$	$\{0.1, 0.001\}$	$\{10/20, 1\}$
	Split CIFAR-100	SGD	$\{0.1, 0.001\}$	$\{30/60/70, 10\}$
		SGD†, $k = 10$	$\{0.1, 0.001\}$	$\{30/60/70, 10\}$
		SGD†, $k = 20$	$\{0.1, 0.001\}$	$\{30/60/70, 10\}$
ViT	Split CIFAR-10	SGD	$\{0.1, 0.001\}$	$\{10/20, 1\}$
		SGD†, $k = 10$	$\{0.1, 0.001\}$	$\{10/20, 1\}$
		SGD†, $k = 20$	$\{0.1, 0.001\}$	$\{10/20, 1\}$
	Split CIFAR-100	SGD	$\{0.1, 0.001\}$	$\{15/25, 5/10\}$
		SGD†, $k = 10$	$\{0.1, 0.001\}$	$\{15/25, 5/10\}$
		SGD†, $k = 20$	$\{0.1, 0.001\}$	$\{15/25, 5/10\}$

Table 4: All the experiments in the table are repeated over 5 seeds, namely $11, 13, 21, 33, 55$. When the same learning rate is used for all the tasks it is written as a single number. Otherwise, we use a different learning rate for the tasks following the first and we indicate this as $\{r_1, r_{2:T}\}$ in the table. The batch size is 10 unless stated otherwise. The epoch number follows a similar scheme: if a different number of epochs is used between the first and remaining tasks we write $\{e_1, e_{2:T}\}$. Moreover, if the learning rate is decayed while learning a task we write the decay schedule in the place of the epoch number as $\{s_1^1/s_1^2/\ldots/e_1, s_{2:T}^1/s_{2:T}^2/\ldots/e_{2:T}\}$, which means that the first learning learning rate is decayed after s_1^1 epochs and then again s_1^2, until reaching the final epoch e_1. Similarly, for the remaining tasks, the learning rate is decayed after $s_{2:T}^1$ epochs and then again $s_{2:T}^2$, and so on until reaching the final epoch $e_{2:T}$.

$1 - \epsilon$ of the total energy. In this way, the size of the 'protected subspace' depends on the task, and in particular on the geometry of the optimum.

B.0.2 Hessian computation and eigendecomposition

The loss Hessian matrix plays a key role in our theoretical framework, and thus its computation is necessary to our experiments. However, the Hessian matrix has a high memory cost, i.e. $\mathcal{O}(P^2)$ (P being the size of the network). Moreover, SGD† uses the principal eigenvectors of the Hessian, and the eigendecomposition operation has complexity $\mathcal{O}(P^3)$. For the MLP network we can compute the full Hessian matrix in $\approx 160s$ and its eigendecomposition in $\approx 960s$ For the CNN and the Transformer networks, we cannot compute the full Hessian matrix. However, we can still obtain its principal eigenvectors through iterative methods such as *power iteration* or the *Lanczos method* and the Hessian-vector product (Yao et al., 2018a; Xu et al., 2018). Golmant et al. (2018) has publicly provided an implementation of these methods for Pytorch neural networks, which we have adapted to our scope. We use a random subset of the data to compute the Hessian and/or its eigenvectors. For all the experiments we fix the subset size to 1000 samples.

C ADDITIONAL RESULTS

C.1 EXTENDED RESULTS

In this section we present the results of the experiments in their entirety. We adopt the same structure of Section 5 of the paper.

C.1.1 EXPERIMENTS SECTION 5.2

With the first set of experiments, we investigate the empirical validity of Theorems 3-4. In the paper, we show the equivalence of GPM, OGD and SGD† when the analysis assumptions are met. Here we further examine GPM and OGD.

Measuring the violation of the constraint Corollary 1 and 2 establish zero forgetting guarantees for OGD and GPM under Assumption 1-2. We directly measure the degree of violation of the null-forgetting constraint (Equation 4) on the RMNIST dataset for both methods and compare them with vanilla SGD. The score used is simply:

$$VNC(t) = \boldsymbol{\Delta}_t^{\intercal}(\frac{1}{t}\sum_{o=1}^{t-1}\mathbf{H}_o^{\star})\boldsymbol{\Delta}_t$$

The null-forgetting constraint is satisfied if $VNC(t) = 0$ for all t. The results, reported in Table 5, agree with our theoretical intuition: GPM and OGD consisently achieve lower degrees of violation of the null-forgetting constraint.

Table 5: Violation of null-forgetting constraint (Equation 4).

lr	Algorithm	$VNC(t)$
$1e^{-5}$	GPM	0.004 ± 0.004
	OGD	$\mathbf{0.0001} \pm 0.0004$
	SGD	0.27 ± 0.10
$1e^{-2}$	GPM	$\mathbf{0.14} \pm 0.11$
	OGD	0.25 ± 0.05
	SGD	0.59 ± 0.07

Verifying the underlying assumptions Next, we check the underlying assumptions of both GPM and OGD. Through our analysis we show that GPM relies on a block-diagonal approximation of the loss Hessian matrix. In order to evaluate the validity of the assumption we define a *block-diagonality* score:

$$\mathfrak{D}(M) = 1 - \frac{\mathbb{E}_{(i,j)\notin D}\,|M_{ij}|}{\mathbb{E}_{(i,j)\in D}\,|M_{ij}|},$$

where we use B to denote the set of entries in the diagonal blocks, and M is any matrix. Notice that a perfectly block-diagonal matrix will score $\mathfrak{D}(M) = 1$. For non block-diagonal matrices the score has no lower bound. On random matrices, the score evaluates to -0.001 (± 0.02), over 1000 trials. In Table 6 (left), we record the average score for the Hessian matrix $\boldsymbol{H}_t^{\star}$ of different tasks when training with GPM, both in the high and low learning rate cases. The average is taken over seeds and tasks. Importantly, the results reveal that the assumption is not met in practice.

OGD effectively adopts the outer-product Hessian $\boldsymbol{H}_t^{O}$ (Equation 9), which has been shown to be a good approximation of the Hessian matrix towards the end of training (Singh et al., 2021) - and thus is automatically satisfied under Assumption 1. We verify this assumption by computing the *spectral similarity* of the Hessian and its outer-product component. We define the spectral similarity between two matrices as follows:

$$\mathfrak{S}(\boldsymbol{A}, \boldsymbol{B}) = \frac{\langle \boldsymbol{A}, \boldsymbol{B}\rangle_F}{\|\boldsymbol{A}\|_F \|\boldsymbol{B}\|_F},$$

where $\langle \cdot, \cdot \rangle_F$ denotes the Frobenius inner product and $\| \cdot \|_F$ denotes the Frobenius norm. This score is equal to 1 for two equivalent matrices, while, for dissimilar matrices it can be arbitrarily low. We

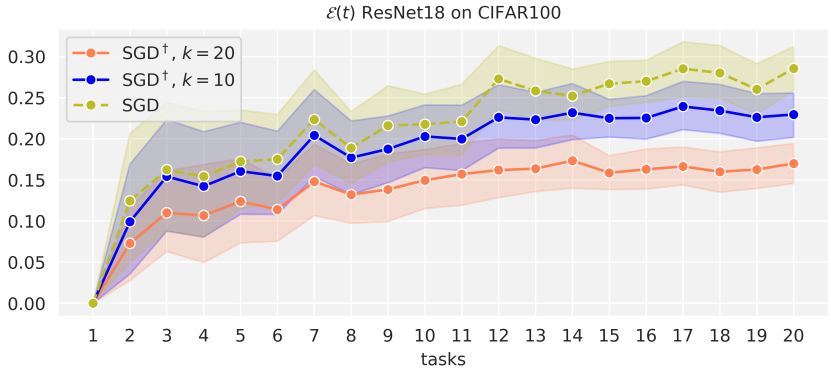

Figure 6: *Convolutional networks*: Average forgetting $\mathcal{E}(t)$ of SGD and SGD† with varying values of k on Split CIFAR-100 for ResNet-18.

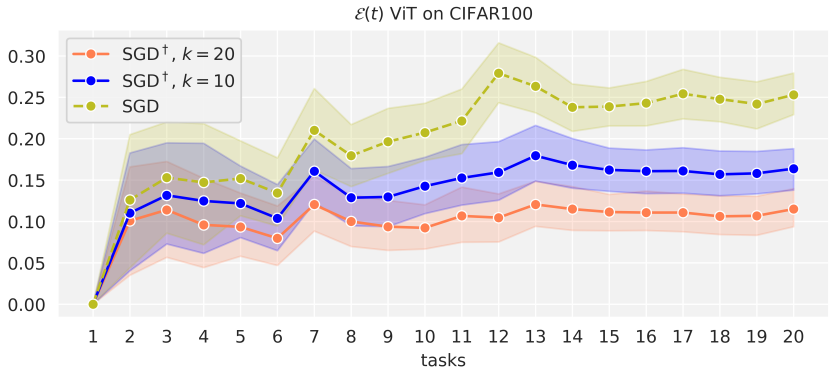

Figure 7: *Vision Transformers*: Average forgetting $\mathcal{E}(t)$ of SGD and SGD† with varying values of k on Split CIFAR-100 for Vision Transformer.

test the score on pairs of random matrices, and obtain an average of $-0.0002\,(\pm 0.01)$ over 1000 trials. In Table 6 (right) we report the score $\mathfrak{S}(\boldsymbol{H}_t^O, \boldsymbol{H}_t^\star)$, averaged over tasks and seeds. For both the high and low learning rate case the score is around 0.7.

Table 6: Validity of the assumptions underlying GPM (left) and OGD (right).

lr	$\mathfrak{D}(\boldsymbol{H}_t^\star)$
$1e^{-5}$	-0.04 ± 0.12
$1e^{-2}$	-0.11 ± 0.13

lr	$\mathfrak{S}(\boldsymbol{H}_t^O, \boldsymbol{H}_t^\star)$
$1e^{-5}$	0.70 ± 0.08
$1e^{-2}$	0.69 ± 0.02

C.1.2 Experiments Section 5.3

The second set of experiments is aimed at testing the validity of our analysis on other architectures. We employ CNNs and Transformers, as described in Section B. Figure 6 and 7 show the average forgetting on Split CIFAR-100 for tasks 1 to 20 (previous results only considered the first 10 tasks for brevity). Moreover, we could compute up to the first 40 eigenvectors for the ResNet, and we include the results in Figure 6.

Next, we report a comprehensive view of the results on Split CIFAR-10 in Table 7. Similarly to Split CIFAR-100, we observe the average forgetting increasing overall with t. Importantly, both the CNN and the Transformer record significantly lower levels of forgetting when trained with SGD†, once again supporting the validity of our theoretical results.

Table 7: $\mathcal{E}(t)$ on Split CIFAR-10.

	Algorithm	$\mathcal{E}(2)$	$\mathcal{E}(3)$	$\mathcal{E}(4)$	$\mathcal{E}(5)$
ResNet	SGD	0.0127 ± 0.0200	0.0186 ± 0.0196	0.0375 ± 0.0515	0.0285 ± 0.0267
	SGD†-10	0.0100 ± 0.0177	0.0092 ± 0.0145	0.0201 ± 0.0198	0.0261 ± 0.0313
	SGD†-20	$\mathbf{0.0094} \pm 0.0169$	$\mathbf{0.0076} \pm 0.0167$	$\mathbf{0.0171} \pm 0.0172$	$\mathbf{0.0213} \pm 0.0262$
ViT	SGD	0.0306 ± 0.0329	0.0249 ± 0.0383	0.0476 ± 0.0669	0.0264 ± 0.0282
	SGD†-10	$\mathbf{0.0122} \pm 0.0153$	0.0073 ± 0.0147	0.0081 ± 0.0161	0.0057 ± 0.0118
	SGD†-20	0.0126 ± 0.0173	$\mathbf{0.0061} \pm 0.0160$	$\mathbf{0.0077} \pm 0.0154$	$\mathbf{0.0048} \pm 0.0110$

Additionally, we report average accuracy on both Split CIFAR-10 (Table 8) and Split CIFAR-100 (Table 9).

Table 8: Average accuracy $a(t)$ on Split CIFAR-10. We limit the learning of each task after the first one to 1 epoch.

	Algorithm	$a(1)$	$a(2)$	$a(3)$	$a(4)$	$a(5)$
ResNet	SGD	0.942 ± 0.01	0.777 ± 0.15	0.714 ± 0.14	0.654 ± 0.11	0.685 ± 0.16
	SGD†-10	0.942 ± 0.01	0.778 ± 0.15	0.713 ± 0.16	0.656 ± 0.16	0.675 ± 0.17
	SGD†-20	0.942 ± 0.01	$\mathbf{0.779} \pm 0.16$	$\mathbf{0.720} \pm 0.15$	$\mathbf{0.664} \pm 0.15$	$\mathbf{0.684} \pm 0.17$
ViT	SGD	0.903 ± 0.003	0.766 ± 0.08	0.739 ± 0.07	0.693 ± 0.04	0.7233 ± 0.10
	SGD†-10	0.903 ± 0.003	$\mathbf{0.785} \pm 0.09$	$\mathbf{0.751} \pm 0.09$	$\mathbf{0.713} \pm 0.10$	$\mathbf{0.7264} \pm 0.09$
	SGD†-20	0.903 ± 0.003	0.783 ± 0.10	0.7480 ± 0.09	0.705 ± 0.11	0.719 ± 0.10

Table 9: Final average accuracy $a(T)$ on Split CIFAR-100, compared to the average current task accuracy $a_{t,t}$ across tasks. We limit the learning of each task after the first one to 10 epochs.

	Algorithm	$a(T)$	$a_{t,t}$
ResNet	SGD	0.2215 ± 0.09	$\mathbf{0.5068} \pm 0.10$
	SGD†-10	0.2366 ± 0.09	0.4660 ± 0.11
	SGD†-20	0.2487 ± 0.09	0.4185 ± 0.11
	SGD†-40	$\mathbf{0.2542} \pm 0.08$	0.3703 ± 0.12
ViT	SGD	0.2258 ± 0.09	$\mathbf{0.4789} \pm 0.09$
	SGD†-10	0.2452 ± 0.09	0.4089 ± 0.11
	SGD†-20	$\mathbf{0.2577} \pm 0.06$	0.3726 ± 0.12

Importantly, the average accuracy reflects the balance of forgetting and intransigence in the network. Lower forgetting comes at the cost of a lower capacity for learning new tasks. On the contrary, a very plastic network performs well on the current task while performing poorly on the old ones. In Table 9 we compare the accuracy on the current task $a_{t,t}$ to the final average accuracy of the network. SGD always outperforms SGD† on the current task, however it achieves the lowest average accuracy. This compromise between forgetting and intransigence, or equivalently between plasticity and stability, is better portrayed in Figure 8, where we show the average accuracy and forgetting of SGD† on Rotated-MNIST as we vary the parameter ϵ, controlling the fraction of spectral energy left of the memory for each task. Simply, $\epsilon = 0$ corresponds to fixing $k = P$, and $\epsilon = 1$ corresponds to fixing $k = 0$. Importantly, the extremes achieve lower average accuracy.

C.1.3 Experiments Section 5.1

The last set of experiments is aimed at assessing the validity of the analysis.

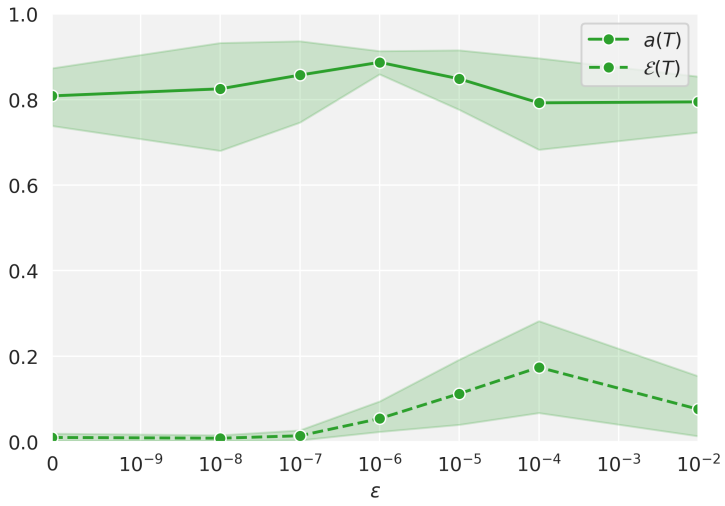

Figure 8: Average final accuracy $a(T)$ and forgetting $\mathcal{E}(T)$ of SGD† on Rotated-MNIST varying ϵ. For each task, a $(1-\epsilon)$ fraction of the spectral energy of the Hessian matrix is stored in memory, thus through ϵ we dynamically control the memory size.

Approximations of forgetting First, we evaluate the quality of two approximations of forgetting, namely a second order Taylor approximation (Equation 2) and the approximation derived from Assumptions 1-2 (Equation 3). Table 10 extends the results shown in Table 1 to the case $o = t-1$, which isolates the contribution of a single update to forgetting. We observe the same trend in both cases, with the second order term in the Taylor expansion providing a stable estimate of forgetting across a wider region of the parameter space. In Table 11 we report the error of the average forgetting

Table 10: Approximation error, Equation 2, on Rotated-MNIST.

	lr	1st order	2nd order	Taylor	$\mathcal{E}_o(t)$	$\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_o\|$
$o \in [1, t-1]$	$1e^{-5}$	0.43 ± 0.30	0.26 ± 0.16	$\mathbf{0.15} \pm 0.15$	0.30 ± 0.30	1.23 ± 0.51
	$1e^{-2}$	2.33 ± 1.87	$\mathbf{1.49} \pm 1.31$	1.55 ± 1.32	2.26 ± 1.85	8.19 ± 2.56
$o = t-1$	$1e^{-5}$	0.16 ± 0.07	0.19 ± 0.09	$\mathbf{0.05} \pm 0.02$	0.018 ± 0.02	0.71 ± 0.13
	$1e^{-2}$	0.39 ± 0.06	$\mathbf{0.09} \pm 0.05$	0.13 ± 0.05	0.33 ± 0.05	5.48 ± 0.17

estimate given by Equation 3 on Rotated MNIST for different values of t. Interestingly, we notice that the error is not monotonic, although forgetting constantly increases.

Table 11: Equation 3 approximation error on Rotated-MNIST.

t	lr	$\lvert\hat{\mathcal{E}}(t) - \mathcal{E}(t)\rvert$	$\mathcal{E}(t)$
2	$1e^{-5}$	$\mathbf{0.047} \pm 0.01$	0.024 ± 0.01
	$1e^{-2}$	0.062 ± 0.07	0.36 ± 0.06
3	$1e^{-5}$	$\mathbf{0.049} \pm 0.005$	0.122 ± 0.02
	$1e^{-2}$	0.74 ± 0.1050	1.27 ± 0.06
4	$1e^{-5}$	$\mathbf{0.073} \pm 0.02$	0.255 ± 0.03
	$1e^{-2}$	1.17 ± 0.24	2.45 ± 0.23
5	$1e^{-5}$	$\mathbf{0.068} \pm 0.05$	0.48 ± 0.07
	$1e^{-2}$	0.80 ± 0.13	3.09 ± 0.14

Finally, we evaluate the quality of the estimates of forgetting for the ResNet experiments on Split CIFAR-100. In order to do so, we employ low rank approximations of the Hessian matrix, obtained through iterative methods. In Table 12 and 13 we report results for multiple rank values, $r = 10, 20, 30, 40$. Overall, the forgetting approximation quality increases with the rank.

Table 12: Approximation error, Equation 2, on Split CIFAR-100.

r	1st order	2nd order	Taylor	$\mathcal{E}_o(t)$	$\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_o\|$
10	0.63 ± 0.31	0.54 ± 0.38	0.60 ± 0.30	0.57 ± 0.39	2.15 ± 0.89
20	0.63 ± 0.31	0.50 ± 0.37	0.54 ± 0.30	0.57 ± 0.39	2.15 ± 0.89
30	0.63 ± 0.31	0.41 ± 0.30	0.42 ± 0.26	0.57 ± 0.39	2.15 ± 0.89
40	0.63 ± 0.31	$\mathbf{0.27} \pm 0.20$	$\mathbf{0.21} \pm 0.16$	0.57 ± 0.39	2.15 ± 0.89

Table 13: Equation 3 approximation error on Split CIFAR-100.

r	$\|\hat{\mathcal{E}}(t) - \mathcal{E}(t)\|$	$\mathcal{E}(t)$
10	0.37 ± 0.17	0.57 ± 0.13
20	0.36 ± 0.16	0.57 ± 0.13
30	0.34 ± 0.14	0.57 ± 0.13
40	$\mathbf{0.30} \pm 0.11$	0.57 ± 0.13

Perturbation analysis In Section 5.1 we propose a tool to measure the range of validity of the assumptions of our analysis, named *perturbation analysis*. Here, we present the detailed results of the perturbation analysis, for all the tasks in the Rotated MNIST setting. These can be found in Figure 9.

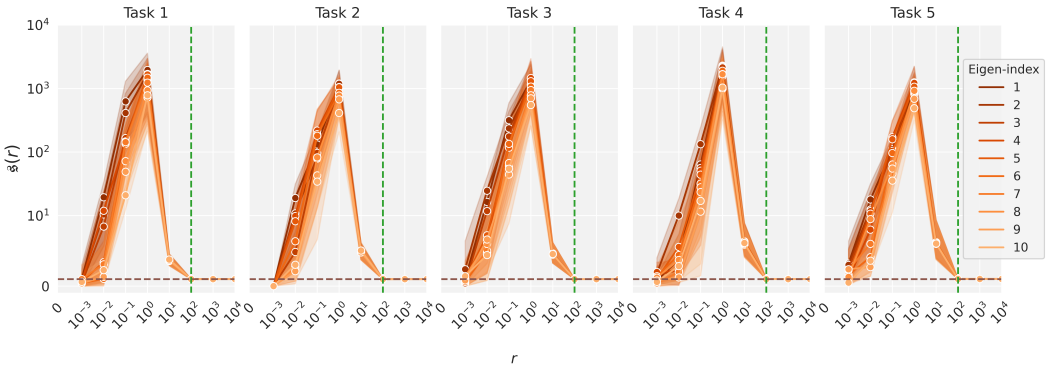

Figure 9: Perturbation score $\mathfrak{s}(r)$ on all the tasks of the Rotated-MNIST challenge, learned with SGD. The dashed green line highlights the value of r for which the score converges to 1 (marked by the dashed brown line).

Next, to further complement these results, we additionally provide results for ResNet18 in the Split-CIFAR100 setting in Figure 10 and that for ViT in Figure 11.

The observations from our perturbation analysis on Rotated-MNIST largely extend to these latter settings. Notice that, as the network size increases, the perturbation score converges to 1 for higher values of r (marked by the dashed green line). There is more noise in some of the plots, which can be attributed to the fact that for these bigger networks, we are relying on the power method to obtain the top eigenvectors, unlike the exact Hessian computation in the case of Rotated-MNIST.

Figure 10: Perturbation score $\mathfrak{s}(r)$ on all the tasks of the Split CIFAR-100 challenge, learned with SGD. The dashed green line highlights the value of r for which the score converges to 1 (marked by the dashed brown line).

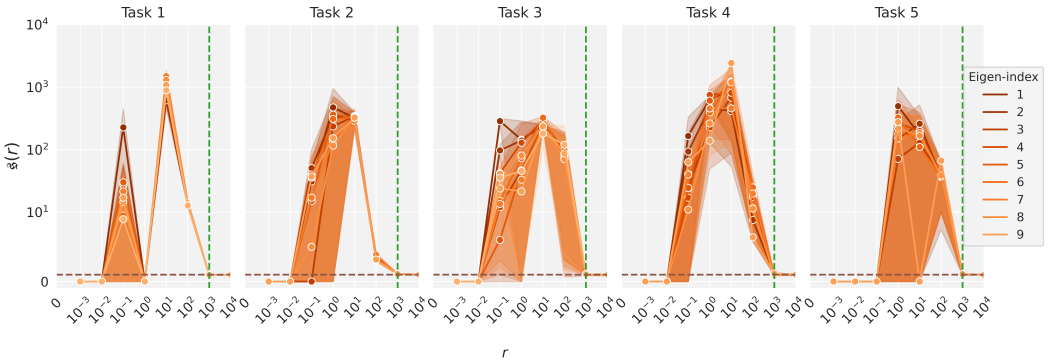

Figure 11: Perturbation score $\mathfrak{s}(r)$ on all the tasks of the Split CIFAR-10 challenge, learned with SGD. The dashed green line highlights the value of r for which the score converges to 1 (marked by the dashed brown line).