Supplementary Material

A. Additional experiments

In Figure 8 we plot the loss function along a line joining the original model trained on all data and the ground-truth optimal model that was trained from the beginning without seeing the data to forget. The plots confirm that the two models are close to each other, as expected from the stability of SGD, and that the loss function in a neighborhood of the two points is convex (at least on the line joining the two). This justifies the hypotheses that we made to apply our forgetting method (derived in the case of a convex loss) to the more challenging case of deep networks.

To further confirm that our method can be applied to different architectures, in addition to the experiments on All-CNN that we show in the main paper we run additional experiments on a ResNet-18 architecture. In particular, Table 2 show the errors obtained by various forgetting techniques and Figure 10 shows the relearn time for a class after forgetting. In Figure 9, we show that ResNets too suffer from a Streisand effect if an improper forgetting procedure is applied.

B. Implementation details

The term $e^{-Bt}e^{At}$ in eq. (6) may diverge for $t \to \infty$ if the actual dynamics of the optimization algorithm do not exactly satisfy the hypotheses. In practice, we found it useful to replace it with $e^{\text{clamp}(A-B,1/t)t}$, where $\text{clamp}(A, 1/t)$ clamps the eigenvalues of A so that they are smaller or equal to $1/t$, so that the expression does not diverge.[2] Notice also that in general $e^{-Bt}e^{At} \neq e^{(A-B)t}$, unless A and B commute, but we found this to be a good approximation in practice.

For DNNs, we perform experiments on Lacuna-10 and CIFAR-10 using All-CNN and ResNet. For All-CNN, we use the model proposed in [27] and for ResNet we use the ResNet-18 model; however, we reduce the number of filters by half in each layer and remove the final residual block. For all the experiments we first pre-train the network on Lacuna-100 (CIFAR-100), and then fine-tune it on Lacuna-10 (CIFAR-10). We do not use data augmentation in any of the experiments. While pre-training the network we use a constant learning rate of 0.1 and train for 30 epochs and while finetuning we use a constant learning rate of 0.01 and train for 50 epochs. In all the experiments we use weight decay regularization with value 0.0005. We use PyTorch to perform all the experiments.

In Table 1, 'Original' model denotes the case when we train the model on the entire dataset, $\mathcal{D}$. 'Retrain' denotes the case when we train on $\mathcal{D}_r$ (we can do this by simply replacing the corresponding samples from the data loader). 'Finetune' is a possible scrubbing procedure when we use the Original model and fine-tune it on $\mathcal{D}_r$ (we can simply use the data loader from the Retrain case for fine-tuning). We run fine-tuning for 10 epochs with a learning rate 0.01 and weight decay 0.0005. 'Negative Gradient' denotes the case when we fine-tune on the complete data; however, for the samples belonging to $\mathcal{D}_f$ we simply multiply the gradient by -1 (i.e. maximizing the cross-entropy loss for the samples to forget). To prevent the loss from diverging, we clamp the maximum loss to chance level. We train for 10 epochs with learning rate 0.01 and weight decay 0.0005. In 'Random Labels,' we randomly shuffle the labels for the samples we want to forget. We use the same training hyperparameters as the previous methods. For 'Hiding,' we replace the row corresponding to the class of the samples to be forgotten in the final classifier layer of the DNN with random initialization. Fisher denotes our method where we estimate the Fisher noise to add to the model. The Fisher noise is computed using a positive semi-definite approximation to the actual Fisher Information Matrix. We compute the trace of the Fisher Information which can be obtained by computing the expected outer product of the gradients of the DNN. In the experiments we observe that $F^{-\frac{1}{2}}$ approximates the Fisher noise better compared to $F^{-\frac{1}{4}}$. In Variational, we compute the optimal noise to be added for scrubbing a cohort by solving a variational problem. For Fisher and Variational forgetting we choose $\lambda = 5 \cdot 10^{-7}$.

In order to compute the Information bound (Fisher and Variational forgetting) we use the Local Forgetting Bound i.e. we apply the scrubbing procedure to both the original and the retrain (target) model and then compute the KL divergence of the two distributions. We use the same random seed while training both the models and use the same scrubbing procedures for both, that is, $S = S_0$. Fisher and Variational forgetting method essentially consists of adding noise to the model weights, i.e., $S(w) = w + n$ where $n \sim \mathcal{N}(0, \Sigma)$. Let w_o, Σ_o and w_r, Σ_r denote the weights and noise covariance for the original and the target retrained model respectively, then the Information bound is given by the following relation:

$$\text{KL}\left(\mathcal{N}(w_o, \Sigma_o) \,\|\, \mathcal{N}(w_r, \Sigma_r)\right) = \frac{1}{2}\Big(\text{tr}(\Sigma_r^{-1}\Sigma_o) + (w_r - w_o)^T \Sigma_r^{-1}(w_r - w_o) - k + \log \frac{|\Sigma_r|}{|\Sigma_o|} \Big),$$

where k is the dimension of w. This bound should be computed for multiple values of the initial random seed and then averaged to obtain the Local Forgetting Bound in Proposition 2. In our experiments we compute the bound using a single random seed.

To compute the Relearn-time we train the scrubbed model on the dataset $\mathcal{D}$ for 50 epochs using a constant learn-

[2] If $A = SDS^T$ is an eigenvalue decomposition of the symmetric matrix A, we define $\text{clamp}(A, m) = S \min(D, m) S^T$.

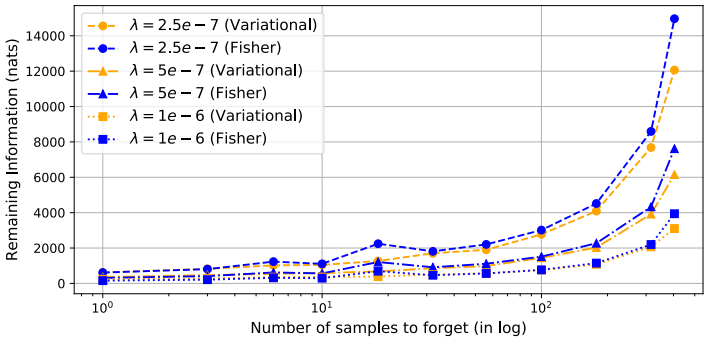

Figure 7. **Difficulty of forgetting increases with cohort size (for ResNet)** We plot the upper-bound on the remaining information (i.e. the information the model contains about the cohort to forget after scrubbing) as a function of the number of samples to forget for class '5' for different values of λ (Forgetting Lagrangian parameter). Increasing the value of λ decreases the remaining information, but increases the error on the remaining samples. The number of samples to forget in the plot varies between one sample and the whole class (404 samples).

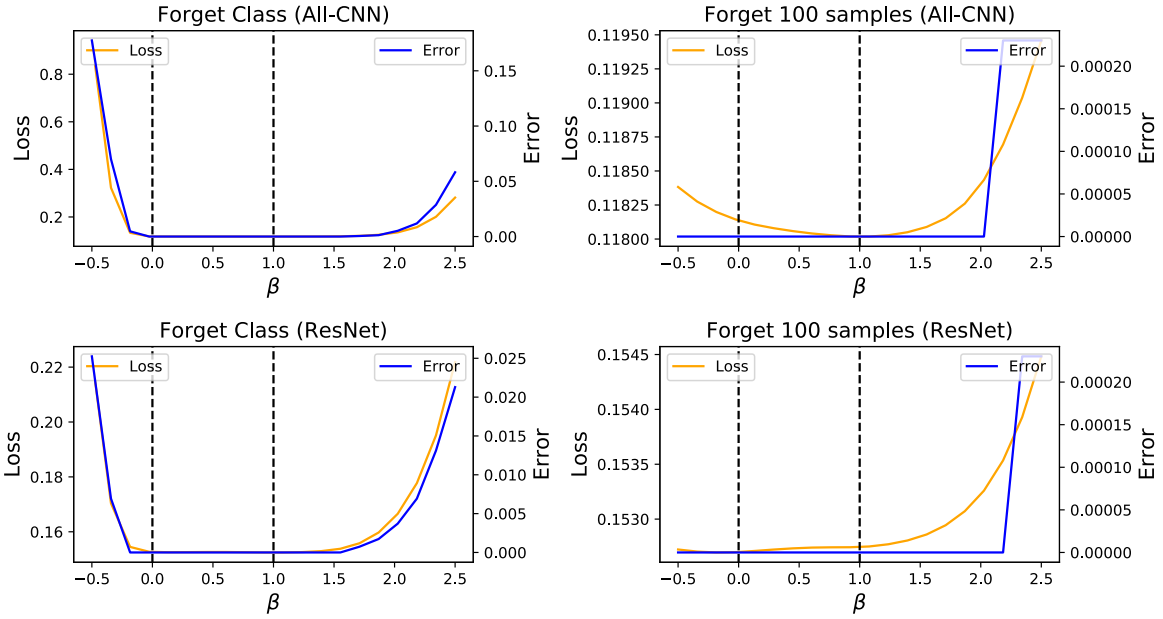

Figure 8. **Loss landscape of $L_{\mathcal{D}_r}$.** Plot of loss and error on the dataset $\mathcal{D}_r$ of the remaining samples, interpolating along the line joining the *original model* (at $t = 0$) and the (target) *retrained model* at $t = 1$. Precisely, let w_o be the original model and let w_r be the retrained model, we plot $L(w(t))$, where $w(t) = (1-t) \cdot w_o + t \cdot w_r$ by varying $t \in [-0.5, 2.5]$. We observe that the loss along the line joining the original and the target model is convex, and almost flat throughout. In particular, there exists at least an optimal direction (the one joining the two models) along which we can add noise without increasing the loss on the remaining data, and which would allow to forget the extra information. This inspires and justifies our forgetting procedure.

ing rate 0.01. We report the first epoch when the loss value falls below a certain threshold as the relearn-time.

B.1. Pre-training improves the forgetting bound

Our local forgetting bound assumes stability of the algorithm, which may not be guaranteed when training a large deep network over many epochs. This can be obviated by pre-training the network, so all paths will start from a common configuration of weights and training time is decreased, and with it the opportunity for paths to diverge. As we show next, the resulting bound is greatly improved. The drawback is that the bound cannot guarantee forgetting of any information contained in the pre-training dataset (that is, $\mathcal{D}_f$ needs to be disjoint from $\mathcal{D}_{\text{pretrain}}$).

B.2. Datasets

We report experiments on MNIST, CIFAR10 [18], Lacuna-10 and Lacuna-100 which we introduce. **Lacuna-**

10 consists of face images of 10 celebrities from VGGFaces2 [6], randomly sampled with at least 500 images each. We split the data into a test set of 100 samples for each class, while the remaining form the training set. Similarly, **Lacuna-100** randomly samples 100 celebrities with at least 500 images each. We resize the images to 32x32. There is no overlap between the two Lacuna datasets. We use Lacuna-100 to pre-train (and hence assume that we do not have to forget it), and fine-tune on Lacuna-10. The scrubbing procedure is required to forget some or all images for one identity in Lacuna-10. On both CIFAR-10 and Lacuna-10 we choose to forget either the entire class '5,' which is chosen at random, or a hundred images of the class.

C. Proofs

Proposition 1 Let the forgetting set $\mathcal{D}_f$ be a random variable, for instance, a random sampling of the data to forget. Let Y be an attribute of interest that depends on $\mathcal{D}_f$. Then,

$$I(Y; f(S(w))) \leq \mathbb{E}_{D_f}[\mathrm{KL}\left(P(f(\mathrm{S}(w))|\mathcal{D}) \,\|\, P(f(\mathrm{S}_0(w))|\mathcal{D}_r)\right)].$$

Proof. We have the following Markov Chain:

$$Y \longleftarrow \mathcal{D}_f \longrightarrow w \longrightarrow S(w) \longrightarrow f(S(w))$$

We assume that the retain set $\mathcal{D}_r$ is fixed and thus deterministic. Also, $\mathcal{D}_f \rightarrow w$ implies that we first sample $\mathcal{D}_f$ and then use $\mathcal{D} = \mathcal{D}_r$(fixed and known) $\cup\, \mathcal{D}_f$ for training w. From Data Processing Inequality, we know that: $I(Y; f(S(w))) \leq I(\mathcal{D}_f; f(S(w)))$. In order to prove the proposition we bound the RHS term (i.e $I(\mathcal{D}_f; f(S(w)))$).

$$\begin{aligned}
&= I(\mathcal{D}_f; f(S(w))) \\
&= \mathbb{E}_{\mathcal{D}_f}\left[\mathrm{KL}\left(p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f(S(w)))\right)\right] \\
&= \mathbb{E}_{\mathcal{D}_f}\mathbb{E}_{p(f(S(w))|\mathcal{D}_f\cup\mathcal{D}_r)}\left[\log \frac{p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r)}{p(f(S(w)))}\right] \\
&= \mathbb{E}_{\mathcal{D}_f}\mathbb{E}_{p(f(S(w))|\mathcal{D}_f\cup\mathcal{D}_r)}\Big[\log \frac{p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r)}{p(f(S_0(w))|\mathcal{D}_r)} \\
&\qquad\qquad + \log \frac{p(f(S_0(w))|\mathcal{D}_r)}{p(f(S(w)))}\Big] \\
&= \mathbb{E}_{\mathcal{D}_f}\left[\mathrm{KL}\left(p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f(S_0(w))|\mathcal{D}_r)\right)\right] \\
&\qquad\qquad - \mathrm{KL}\left(p(f(S_0(w))) \,\|\, p(f(S(w))|\mathcal{D}_r)\right) \\
&\leq \mathbb{E}_{\mathcal{D}_f}\left[\mathrm{KL}\left(p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f(S_0(w))|\mathcal{D}_r)\right)\right]
\end{aligned}$$

where the last inequality follows from the fact that KL-divergence is always non-negative.

□

Lemma 1 For any function $f(w)$ have

$$\mathrm{KL}\left(P(f(\mathrm{S}(w))|\mathcal{D}) \,\|\, P(f(\mathrm{S}_0(w))|\mathcal{D}_r)\right) \leq \mathrm{KL}\left(P(\mathrm{S}(w)|\mathcal{D}) \,\|\, P(\mathrm{S}_0(w)|\mathcal{D}_r)\right),$$

Proof. For simplicity sake, we will consider the random variables to be discrete. To keep the notation uncluttered, we consider the expression

$$\mathrm{KL}\left(Q(f(x)) \,\|\, R(f(x))\right) \leq \mathrm{KL}\left(Q(x) \,\|\, R(x)\right)$$

where $Q(w) = P(\mathrm{S}(w)|\mathcal{D})$ and $R(w) = P(\mathrm{S}_0(w)|\mathcal{D}_{-k})$. Let $W_c = \{w|f(w) = c\}$, so that we have $Q(f(w) = c) = \sum_{w\in W_c} Q(w)$ and similarly $R(f(w) = c) = \sum_{w\in W_c} R(w)$. Rewriting the LHS with this notation, we get:

$$\begin{aligned}
&\mathrm{KL}\left(Q(f(w)) \,\|\, R(f(w))\right) \\
&\quad= \sum_c Q(f(w) = c) \log \frac{Q(f(w) = c)}{R(f(w) = c)} \\
&\quad= \sum_c Q(f(w) = c) \log \frac{\sum_{w\in W_c} Q(w)}{\sum_{w\in W_c} R(w)} \qquad (9)
\end{aligned}$$

We can similarly rewrite the RHS:

$$\mathrm{KL}\left(Q(w) \,\|\, R(w)\right) = \sum_c \sum_{w\in W_c} \log \frac{Q(w)}{R(w)} \qquad (10)$$

From the log-sum inequality, we know that for each c in eq. (9) and eq. (10):

$$\Big(\sum_{w\in W_c} Q(w)\Big) \log \frac{\sum_{w\in W_c} Q(w)}{\sum_{w\in W_c} R(w)} \leq \sum_{w\in W_c} Q(w) \log \frac{Q(w)}{R(w)}$$

Summation over all c on both sides of the inequality concludes the proof. □

Proposition 2 Let $A(\mathcal{D})$ be a (possibly stochastic) training algorithm, the outcome of which we indicate as $w = A(\mathcal{D}, \epsilon)$ for some deterministic function A and $\epsilon \sim \mathcal{N}(0, I)$, for instance a random seed. Then, we have $P(S(w)|\mathcal{D}) = \mathbb{E}_\epsilon[P(S(w)|\mathcal{D}, \epsilon)]$. We have the bound:

$$\mathrm{KL}\left(P(S(w)|\mathcal{D}) \,\|\, P(S_0(w)|\mathcal{D}_r)\right) \leq \mathbb{E}_\epsilon\Big[\mathrm{KL}\left(P(S(w)|\mathcal{D}, \epsilon) \,\|\, P(S_0(w)|\mathcal{D}_r, \epsilon)\right)\Big]$$

Proof. To keep the notation uncluttered,we rewrite the inequality as:

$$\mathrm{KL}\left(Q(w) \,\|\, R(w)\right) \leq \mathbb{E}_\epsilon\Big[\mathrm{KL}\left(Q(w|\epsilon) \,\|\, R(w|\epsilon)\right)\Big]$$

where $Q(w) = P(\mathrm{S}(w)|\mathcal{D})$ and $R(w) = P(\mathrm{S}(w)|\mathcal{D}_r)$ The

LHS can be equivalently written as:

$$\begin{aligned}
&\mathrm{KL}\left(Q(w)\,\|\,R(w)\right) = \\
&\quad = \int Q(w)\log\frac{Q(w)}{R(w)})dw \\
&\quad = \int \mathbb{E}_\epsilon[Q(w|\epsilon)]\log\frac{\mathbb{E}_\epsilon[Q(w|\epsilon)]}{\mathbb{E}_\epsilon[P(w|\epsilon)]}dw \\
&\quad \overset{(*)}{\leq} \int \mathbb{E}_\epsilon\Big[Q(w|\epsilon)\log\frac{Q(w|\epsilon)}{P(w|\epsilon)}\Big]dw \\
&\quad = \mathbb{E}_\epsilon\Big[\int Q(w|\epsilon)\log\frac{Q(w|\epsilon)}{P(w|\epsilon)}dw\Big] \\
&\quad = \mathbb{E}_\epsilon\big[\,\mathrm{KL}\left(Q(w|\epsilon)\,\|\,R(w|\epsilon)\right)\big],
\end{aligned}$$

where in (*) we used the log-sum inequality. □

Proposition 3 Let the loss be $L_{\mathcal{D}}(w) = L_{\mathcal{D}_f}(w) + L_{\mathcal{D}_r}(w)$, and assume both $L_{\mathcal{D}}(w)$ and $L_{\mathcal{D}_r}(w)$ are quadratic. Assume that the optimization algorithm $A_t(\mathcal{D},\epsilon)$ at time t is given by the gradient flow of the loss with a random initialization, and let $h(w)$ be the function:

$$h(w) = e^{-Bt}e^{At}d + e^{-Bt}(d - d_r) - d_r,$$

where $A = \nabla^2 L_{\mathcal{D}}(w)$, $B = \nabla^2 L_{\mathcal{D}_r}(w)$, $d = A^{-1}\nabla_w L_{\mathcal{D}}$ and $d_r = B^{-1}\nabla_w L_{\mathcal{D}_r}$, and e^{At} denotes the matrix exponential. Then $h(A_t(\mathcal{D},\epsilon)) = A_t(\mathcal{D}_r,\epsilon)$ for all random initializations ϵ and all times t.

Proof. Since $L_{\mathcal{D}}$ and $L_{\mathcal{D}_r}$ are quadratic, assume without loss of generality that:

$$\begin{aligned}
L_{\mathcal{D}}(w) &= \frac{1}{2}(w - w_A^*)^T A(w - w_A^*) \\
L_{\mathcal{D}_f}(w) &= \frac{1}{2}(w - w_B^*)^T B(w - w_B^*)
\end{aligned}$$

Since the training dynamic is given by a gradient flow, the training path is the solution to the differential equation:

$$\begin{aligned}
\dot{w_A}(t) &= A(w(t) - w_A^*) \\
\dot{w_B}(t) &= B(w(t) - w_B^*)
\end{aligned}$$

which is given respectively by:

$$\begin{aligned}
w_A(t) &= w_A^* + e^{-At}(w_0 - w_A^*) \\
w_B(t) &= w_B^* + e^{-Bt}(w_0 - w_B^*)
\end{aligned}$$

We can compute w_0 from the first expression:

$$w_0 = e^{At}(w_A(t) - w_A^*) + w_A^* = e^{At}d_A + w_A^*,$$

where we defined $d_A = w_A(t) - w_A^* = A^{-1}\nabla_w L_{\mathcal{D}}(w_A(t))$. We now replace this expression of w_0 in the second expression to obtain:

$$\begin{aligned}
w_B(t) &= e^{-Bt}e^{At}d_A + e^{-Bt}(w_A^* - w_B^*) + w_B^* \\
&= w_A(t) + e^{-Bt}e^{At}d_A + e^{-Bt}(d_B - d_A) - d_B
\end{aligned}$$

where $d_B := w_A(t) - w_B^* = B^{-1}\nabla_w L_{\mathcal{D}_r}(w_A(t))$. □

Proposition 4 Assume that $h(w)$ is close to w' up to some normally distributed error $h(w) - w' \sim N(0,\Sigma_h)$, and assume that $L_{\mathcal{D}_r}(w)$ is (locally) quadratic around $h(w)$. Then the optimal scrubbing procedure in the form $S(w) = h(w) + n, \quad n \sim N(0,\Sigma)$,that minimizes the Forgetting Lagrangian

$$\begin{aligned}
\mathcal{L} = \mathbb{E}_{\tilde{w}\sim S(w)}[L_{\mathcal{D}_r}(\tilde{w})] + \\
\lambda\,\mathbb{E}_\epsilon\Big[\,\mathrm{KL}\left(P(S(w)|\mathcal{D},\epsilon)\,\|\,P(S_0(w)|\mathcal{D}_r,\epsilon)\right)\Big]
\end{aligned}$$

is obtained when $\Sigma B\Sigma = \lambda\Sigma_h$, where $B = \nabla^2 L_{\mathcal{D}_r}(k)$. In particular, if the error is isotropic, that is $\Sigma_h = \sigma_h^2 I$ is a multiple of the identity, we have $\Sigma = \sqrt{\lambda\sigma_h^2}B^{-1/2}$.

Proof. We consider the following second order approximation to the loss function in the neighbourhood of the parameters at convergence:

$$\begin{aligned}
&\mathbb{E}_{\tilde{w}\sim S(w)}[L_{\mathcal{D}_r}(\tilde{w})] = \mathbb{E}_{n\sim N(0,\Sigma)}[L_{\mathcal{D}_r}(h(w)+n)] \\
&\quad = \mathbb{E}_{n\sim N(0,\Sigma)}\Big[L_{\mathcal{D}_r}(h(w)) + \\
&\qquad \nabla L_{\mathcal{D}_r}(h(w))^T\Sigma^{\frac{1}{2}}n + \frac{1}{2}(\Sigma^{\frac{1}{2}}n)^T B(\Sigma^{\frac{1}{2}}n) + o(n^2)\Big] \\
&\quad \simeq L_{\mathcal{D}_r}(h(w)) + \frac{1}{2}\mathbb{E}_{n\sim N(0,\Sigma)}\Big[(\Sigma^{\frac{1}{2}}n)^T B(\Sigma^{\frac{1}{2}}n)\Big] \\
&\quad = L_{\mathcal{D}_r}(h(w)) + \frac{1}{2}\mathbb{E}_{n\sim N(0,\Sigma)}\Big[\,\mathrm{tr}(B(\Sigma^{\frac{1}{2}}nn^T\Sigma^{\frac{1}{2}}))\Big] \\
&\quad = L_{\mathcal{D}_r}(h(w)) + \frac{1}{2}\,\mathrm{tr}(B\Sigma)
\end{aligned}$$

Recall that we take S_0 to be $S_0(w) = w + n'$ with $n' \sim N(0,\Sigma)$, so that $P(S_0(w)|\mathcal{D}_r,\epsilon) \sim N(w,\Sigma)$. Thus, we get the following expression for the KL divergence term:

$$\begin{aligned}
&\mathbb{E}_\epsilon\Big[\,\mathrm{KL}\left(P(S(w)|\mathcal{D},\epsilon)\,\|\,P(S_0(w)|\mathcal{D}_r,\epsilon)\right)\Big] \\
&\quad = \frac{1}{2}\mathbb{E}_\epsilon\Big[(h(w) - w')^T\Sigma^{-1}(h(w) - h(w'))\Big] \\
&\quad = \frac{1}{2}\,\mathrm{tr}(\Sigma^{-1}\Sigma_h),
\end{aligned}$$

where in the last equality we have used that by hypothesis $h(w) - w' \sim N(0,\Sigma_h)$. Combining the two terms we get:

$$\mathcal{L} = L_{\mathcal{D}_r}(h(w)) + \frac{1}{2}\,\mathrm{tr}(B\Sigma) + \frac{1}{2}\,\mathrm{tr}(\Sigma^{-1}\Sigma_h)$$

We now want to find the optimal covariance Σ of the noise to add in order to forget. Setting $\nabla_\Sigma\mathcal{L} = 0$, we obtain the following optimality condition $\Sigma B\Sigma = \lambda\Sigma_h$. If we further assume the error Σ_h to be isotropic, that is, $\Sigma_h = \sigma_h^2 I$, then this condition simplifies to $\Sigma = \sqrt{\lambda\sigma_h^2}B^{-1/2}$. □

Table 2. **Same experiment as Table 1 but with a different architecture: Error readout functions for ResNet trained on Lacuna-10 and CIFAR-10.**

	Metrics	**Original model**	**Retrain** (target)	**Finetune**	**Neg. Grad.**	**Rand. Lbls.**	**Hiding**	**Fisher** (ours)	**Variational** (ours)
Lacuna-10	Error on $\mathcal{D}_{\text{test}}$ (%)	18.0 ±0.5	18.2 ±0.3	18.1 ±0.2	18.0 ±0.5	19.3 ±0.7	25.1 ±0.1	24.5 ±0.7	28.2 ±3.2
Scrub 100	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	29.0 ±0.1	0.0 ±0.0	0.0 ±0.0	14.3 ±6.7	100 ±0.0	17.7 ±6.7	3.0 ±1.0
ResNet	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.2 ±0.0	6.5 ±0.0	10.4 ±0.8	11.7 ±5.8
	Info-bound (kNATs)							2.6 ±0.1	2.6 ±0.3
Lacuna-10	Error on $\mathcal{D}_{\text{test}}$ (%)	18.0 ±0.5	24.5 ±0.4	18.1 ±0.4	25.0 ±0.4	24.6 ±0.8	22.0 ±5.2	26.6 ±1.0	26.8 ±0.6
Forget class	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	100 ±0.0	0.0 ±0.0	99.7 ±0.3	90.3 ±1.5	100.0 ±0.0	100.0 ±0.0	100.0 ±0.0
ResNet	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.2 ±0.4	1.0 ±0.9
	Info-bound (kNATs)							33.2 ±2.2	25.7 ±7.8
CIFAR-10	Error on $\mathcal{D}_{\text{test}}$ (%)	17.3 ±0.2	17.5 ±1.0	17.3 ±0.5	17.2 ±0.3	17.9 ±1.2	23.6 ±1.3	21.9 ±2.2	23.1 ±2.3
Scrub 100	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	25.2 ±5.2	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	100.0 ±0.0	16.0 ±7.2	13.3 ±7.4
ResNet	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	8.7 ±1.9	7.6 ±4.0	9.8 ±3.9
	Info-bound (kNATs)							46.0 ±19.3	29.8 ±8.0
CIFAR-10	Error on $\mathcal{D}_{\text{test}}$ (%)	17.1 ±0.3	22.8 ±0.0	17.2 ±0.1	22.8 ±0.1	23.1 ±0.2	22.8 ±0.1	25.8 ±0.1	25.1 ±0.3
Forget class	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	100 ±0.0	0.7 ±0.5	100 ±0.1	94.2 ±5.7	100.0 ±0.0	100.0 ±0.0	100.0 ±0.0
ResNet	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	3.8 ±0.4	2.4 ±1.0
	Info-bound (kNATs)							235.4 ±4.9	234.8 ±6.1

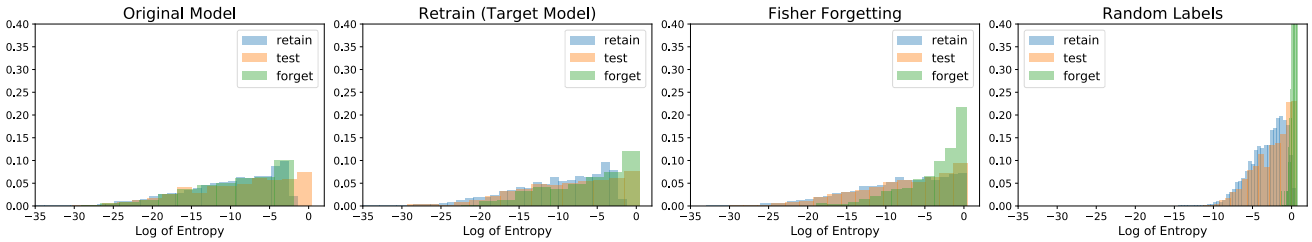

Figure 9. **Same plot as in Figure 4 but with a different architecture: Streisand Effect for ResNet model**

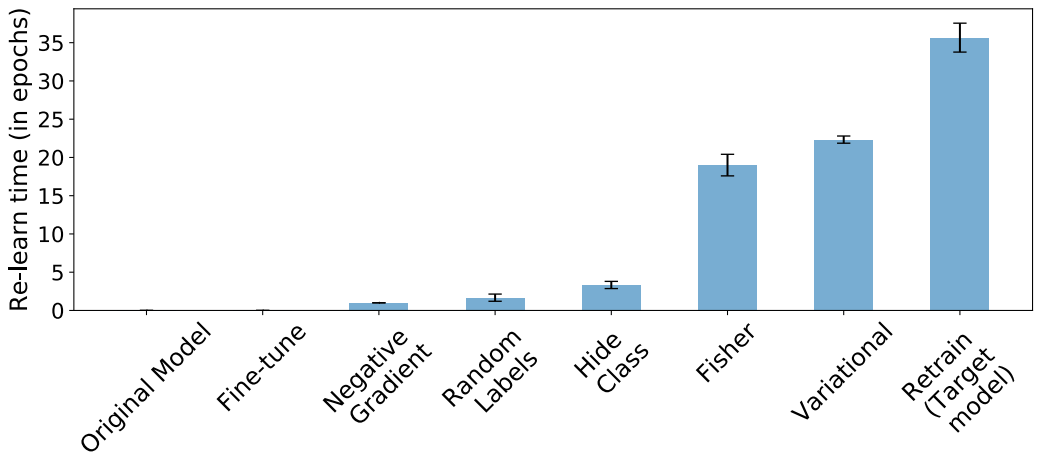

Figure 10. **Same plot as in Figure 5 but with a different architecture: Re-learn time (in epochs) for various forgetting methods using ResNet model.**