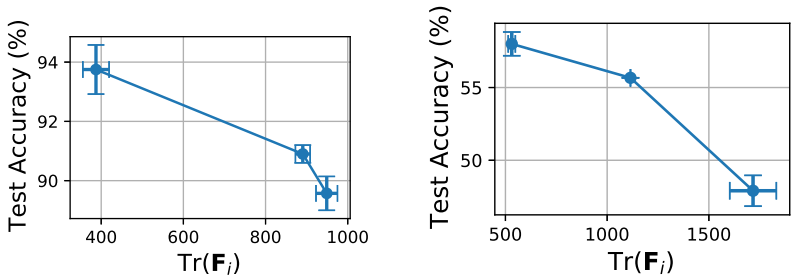
# Apppendix

## A. Additional results

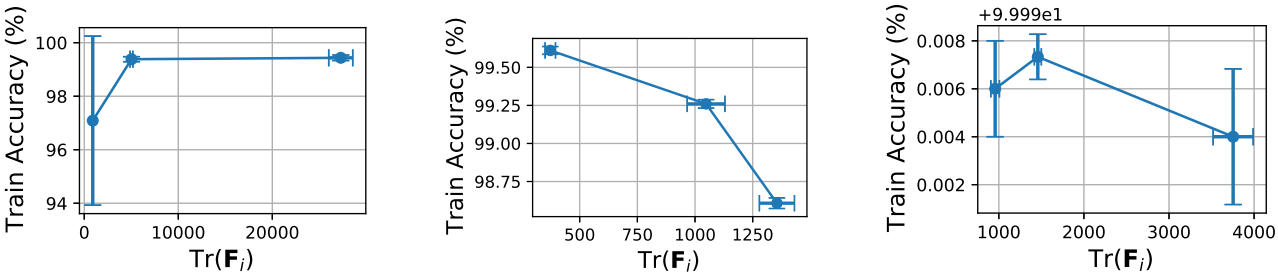
### A.1. Early phase $\mathrm{Tr}(\mathbf{F})$ correlates with final generalization

In this section, we present the additional experimental results for Section 3. Figure 7 shows the experiments with varying batch size for CIFAR-100 and CIFAR-10. The conclusions are the same as discussed in the main text in Section 3. We also show the training accuracy for all the experiments performed in Figure 2 and Figure 7. They are shown in Figure 8 and Figure 9 respectively. Most runs in all these experiments reach training accuracy $\sim$99% and above.



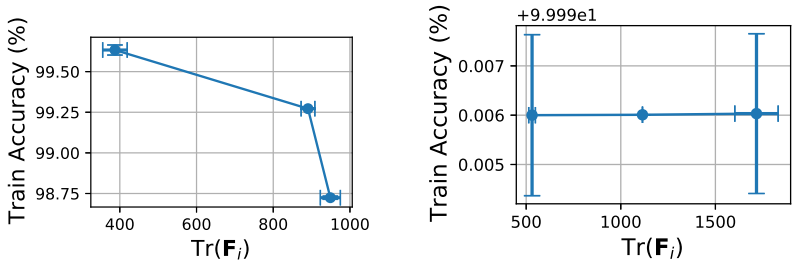

(a) CIFAR-10 (with aug.) (b) CIFAR-100 (w/o aug.)

Figure 7: Association between early phase values of $\mathrm{Tr}(\mathbf{F})$ and generalization holds on the CIFAR-10 and CIFAR-100 datasets. Each point corresponds to multiple runs with randomly chosen seeds and a specific value of batch size. $\mathrm{Tr}\mathbf{F}_i$ is recorded during the early phase (2-7 epochs, see main text for details), while the test accuracy is the maximum value along the entire optimization path (averaged across runs with the same batch size). The horizontal and vertical error bars show the standard deviation of values across runs. The plots show that early phase $\mathrm{Tr}(\mathbf{F})$ is predictive of final generalization.




(a) ImageNet (w/o augmentation) (b) CIFAR-10 (with augmentation) (c) CIFAR-100 (w/o augmentation)

Figure 8: Training accuracy for the runs corresponding to Figure 2. Each point corresponds to multiple seeds and a specific value of learning rate. $\mathrm{Tr}(\mathbf{F_i})$ is recorded during the early phase of training (2-7 epochs, see the main text for details), while the training accuracy is the maximum value along the entire optimization path (averaged across runs with the same learning rate).




(a) CIFAR-10 (with aug.) (b) CIFAR-100 (w/o aug.)

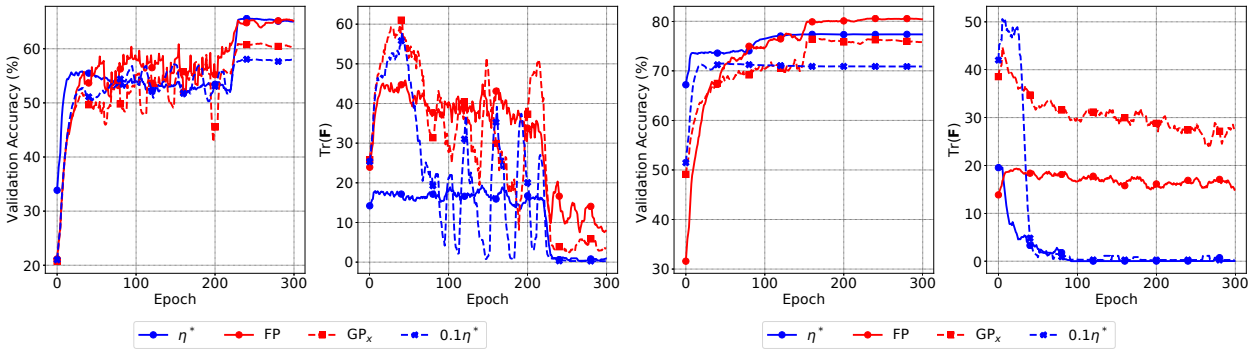
Figure 9: Training accuracy for the runs corresponding to Figure 7. Each point corresponds to multiple runs with randomly chosen seeds and a specific value of batch size. Tr$\mathbf{F}_i$ is recorded during the early phase (2-7 epochs, see main text for details), while the training accuracy is the maximum value along the entire optimization path (averaged across runs with the same batch size).

### A.2. Fisher Penalty

We first show additional metrics for experiments summarized in Table 1 in the main text. Table 6 summarizes the final training accuracy, showing that the baseline experiments were trained until approximately 100% training accuracy was reached. Table 4 supports the claim that all gradient norm regularizers reduce the maximum value of $\mathrm{Tr}(\mathbf{F})$ (we measure $\mathrm{Tr}(\mathbf{F})$ starting from after one epoch of training because $\mathrm{Tr}(\mathbf{F})$ explodes in networks with batch normalization layers at initialization). Finally, Table 5 confirms that the regularizers incurred a relatively small additional computational cost.

Figure 10 complements Figure 4 for the other two models on the CIFAR-10 and CIFAR-100 datasets. The figures are in line with the results of the main text.

Lastly, in Table 7 we report the final training accuracy reached by runs reported in Table 2 in the main text.




(a) DenseNet on CIFAR-100 (w/o aug.) (b) SimpleCNN on CIFAR-10 (w/o aug.)

Figure 10: Same as Figure 4, but for DenseNet on CIFAR-100, and SimpleCNN on CIFAR-10. Curves were smoothed for visual clarity.

Table 4: The maximum value of $\mathrm{Tr}(\mathbf{F})$ along the optimization trajectory for experiments on CIFAR-10 or CIFAR-100 included in Table 1.

| Setting | $\eta^*$ | Baseline | $GP_x$ | GP | FP | $GP_r$ |
|---|---|---|---|---|---|---|
| DenseNet/C100 (w/o aug.) | 24.68 | 98.17 | 83.64 | 64.33 | 66.24 | 73.66 |
| VGG11/C100 (w/o aug.) | 50.88 | 148.19 | 102.95 | 58.53 | 64.93 | 62.96 |
| WResNet/C100 (w/o aug.) | 26.21 | 91.39 | 41.43 | 40.94 | 56.53 | 39.31 |
| SCNN/C10 (w/o aug.) | 24.21 | 52.05 | 47.96 | 25.03 | 19.63 | 25.35 |

Table 5: Time per epoch (in seconds) for experiments in Table 1.

| Setting | $\eta^*$ | Baseline | $GP_x$ | GP | FP | $GP_r$ |
|---|---|---|---|---|---|---|
| WResNet/TinyImageNet (aug.) | 214.45 | 142.69 | 233.14 | 143.78 | 208.62 | 371.74 |
| DenseNet/C100 (w/o aug.) | 78.88 | 57.40 | 77.89 | 78.66 | 97.25 | 75.96 |
| VGG11/C100 (w/o aug.) | 30.50 | 35.27 | 31.54 | 32.52 | 43.41 | 42.40 |
| WResNet/C100 (w/o aug.) | 49.64 | 47.99 | 71.33 | 61.36 | 76.93 | 53.25 |
| SCNN/C10 (w/o aug.) | 18.64 | 19.51 | 26.09 | 19.91 | 21.21 | 20.55 |

Table 6: The final epoch training accuracy for experiments shown in Table 1. Experiments with small learning rate reach no lower accuracy than experiments corresponding to a large learning rate $\eta^*$.

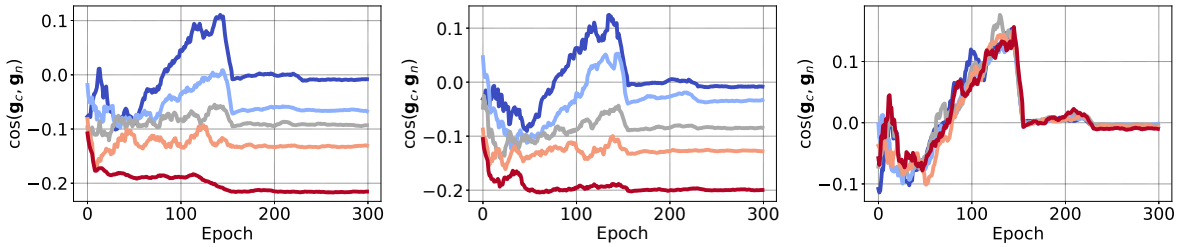
| Setting | $\eta^*$ | Baseline | $GP_x$ | GP | FP | $GP_r$ |
|---|---|---|---|---|---|---|
| WResNet/TinyImageNet (aug.) | 99.84% | 99.96% | 99.97% | 93.84% | 81.05% | 86.46% |
| DenseNet/C100 (w/o aug.) | 99.98% | 99.97% | 99.96% | 99.91% | 99.91% | 99.39% |
| VGG11/C100 (w/o aug.) | 99.98% | 99.98% | 99.85% | 99.62% | 97.73% | 86.32% |
| WResNet/C100 (w/o aug.) | 99.98% | 99.98% | 99.97% | 99.96% | 99.99% | 99.94% |
| SCNN/C10 (w/o aug.) | 100.00% | 100.00% | 97.79% | 100.00% | 93.80% | 94.64% |

Table 7: The final epoch training accuracy for experiments shown in Table 2.

| Setting | $\eta^*$ | FP |
|---|---|---|
| DenseNet/C100 (aug) | 98.75±0.27% | 97.53±1.75% |
| SCNN/C10 (aug) | 97.52±1.98% | 98.94±0.08% |
| VGG11/C100 (aug) | 98.64±0.06% | 93.06±0.01% |
| WResNet/C100 (aug) | 99.97±0.01% | 99.97±0.00% |
| WResNet/Tiny ImageNet (aug) | 99.86±0.02% | 93.65±5.85% |

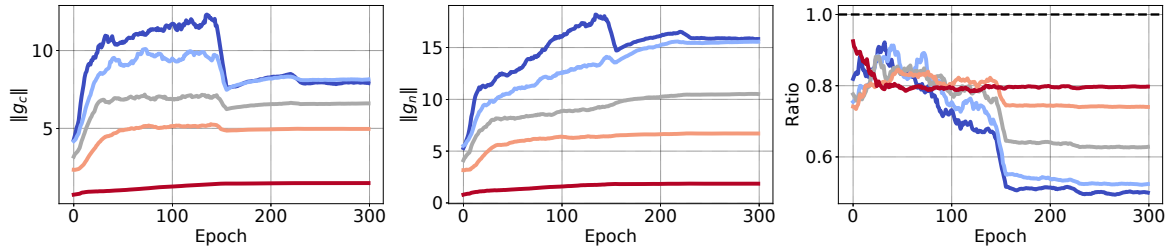
### A.3. Fisher Penalty Reduces Memorization

In this section, we include additional experimental results for Section 4.1. Figure 11 is the same as Figure 5, but for ResNet-50. Finally, we show additional metrics for the experiments involving 25% noisy examples. Figure 12 shows the cosine between the mini-batch gradients computed on the noisy and clean data. In Table 9 and Table 8 we show training accuracy on the noisy and clean examples in the final epoch of training.

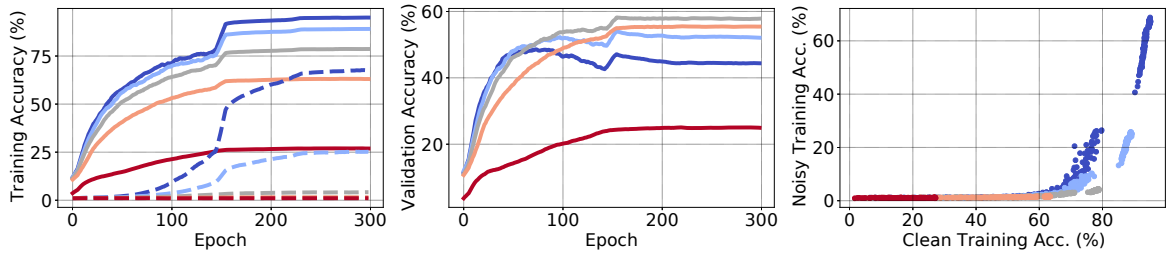



(a) $GP_r$ (b) FP (c) $GP_x$

Figure 12: The cosine between the mini-batch gradients computed on the noisy ($\mathbf{g}_n$) and clean ($\mathbf{g}_c$) data, both measured on the training set. We observe that in the early phase of training the angle is negative. Furthermore, stronger regularization with $GP_r$ or FP correlates with a more negative angle.




(a) Gradient norm on clean examples (left, denoted as $g_c$), noisy examples (middle, denoted as $g_n$), and their ratio (right); evaluated on the training set.




(b) Training accuracy on clean and noisy examples (solid/dashed lined, left), validation accuracy (middle), and a scatter plot of training accuracy on clean vs noisy examples (right).

Figure 11: Same as Figure 5, but for ResNet-50 trained on the CIFAR-100 dataset.

Table 8: Training accuracy on the clean examples int the final epoch, for experiments reported in Table 3 (with 25% examples with noisy labels).

| Label Noise | Setting | Baseline | Mixup | $GP_x$ | FP | $GP_r$ |
|---|---|---|---|---|---|---|
| 25% | VGG-11/C100 | 99.79% | 73.14% | 97.46% | 79.52% | 81.75% |
| | ResNet-52/C100 | 95.87% | 77.71% | 95.88% | 78.72% | 74.21% |

### A.4. Early $\mathrm{Tr}(\mathbf{F})$ influences final curvature

In this section, we present additional experimental results for Section 5. The experiment on CIFAR-10 is shown in Figure 13. The conclusions are the same as discussed in the main text in Section 5.

Table 9: Training accuracy on the noisy examples in the final epoch, for experiments reported in Table 3 (with 25% examples with noisy labels).

| Label Noise | Setting | Baseline | Mixup | $GP_x$ | FP | $GP_r$ |
|---|---|---|---|---|---|---|
| 25% | VGG-11/C100 | 99.56% | 8.29% | 89.23% | 4.10% | 4.96% |
| | ResNet-52/C100 | 73.67% | 4.22% | 72.67% | 4.14% | 2.86% |

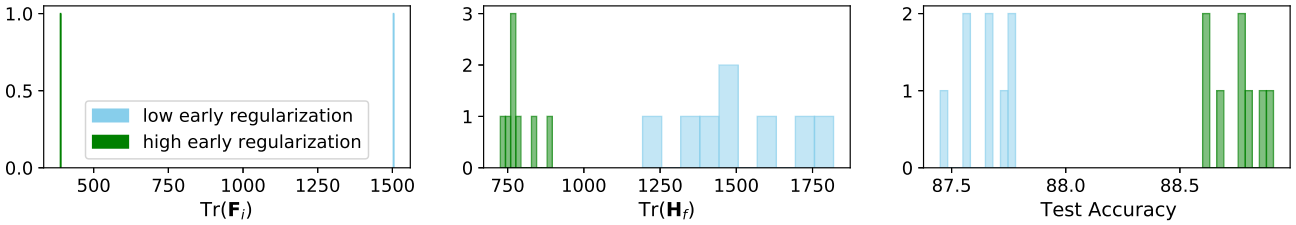
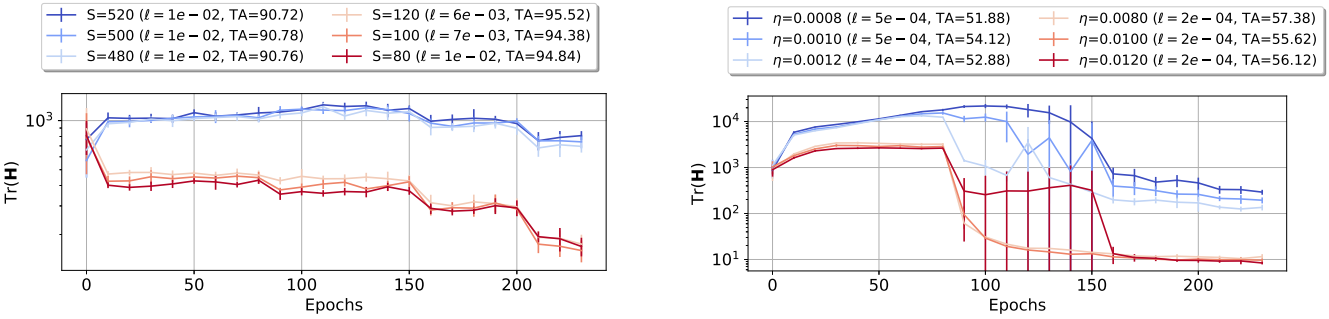



Figure 13: Small $\mathrm{Tr}(\mathbf{F})$ during the early phase of training is more likely to reach wider minima as measured by $\mathrm{Tr}(\mathbf{H})$. Left: 2 models are trained with different levels of regularization for 20 epochs on CIFAR-10. $\mathrm{Tr}(\mathbf{F})$ at the end of 20 epochs (denoted as $\mathrm{Tr}(\mathbf{F_i})$) is shown. Middle: Each model is then used as initialization and trained until convergence using the low regularization configuration with different random seeds. A histogram of $\mathrm{Tr}(\mathbf{H})$ at the point corresponding to the best test accuracy along the trajectory (denoted by $\mathrm{Tr}(\mathbf{H_f})$) is shown. Right: a histogram of the best test accuracy corresponding to middle figure is shown.




(a) CIFAR-10 (w/ augmentation)

(b) CIFAR-100 (w/o augmentation)

Figure 14: The value of $\mathrm{Tr}(\mathbf{H})$ over the course of training. Each point corresponds to runs with different seeds and a specific value of learning rate $\eta$ and batch size $S$. $\ell$ and TA respectively denote the minimum training loss and the maximum test accuracy along the entire trajectory for the corresponding runs (averaged across seeds). The plots show that flatter optimization trajectories become biased towards flatter minima early during training, at a coarse scale of hyper-parameter values (red vs blue).

Next, to understand why smaller $\mathrm{Tr}(\mathbf{F})$ during the early phase is more likely to end up in a wider final minimum, we track $\mathrm{Tr}(\mathbf{H})$ during the entire course of training and show that it stabilizes early on. In this experiment, we create two sets of hyper-parameters: coarse-grained and fine-grained. For CIFAR-10, we use batch size $S \in A \cup B$, where $A = \{480, 500, 520\}$ and $B = \{80, 100, 120\}$. For all batch size configurations, a learning rate of 0.02 is used. Overloading the symbols $A$ and $B$ for CIFAR-100, we use learning rate $\eta \in A \cup B$, where $A = \{0.0008, 0.001, 0.0012\}$ and $B = \{0.008, 0.01, 0.012\}$. For all learning rate configurations, a batch size of 100 is used. In both cases, the elements within each set ($A$ and $B$) vary on a fine-grained scale, while the elements across the two sets vary on a coarse-grained scale. The remaining details and additional experiments can be found in Appendix I.4. The experiments are shown in Figure 14. Notice that after initialization (index 0 on the horizontal axis), the first value is computed at epoch 10 (at which point the experiments show that entanglement has started to hold in alignment with the late phase).

We make three observations in this experiment. First, the relative ordering of $\mathrm{Tr}(\mathbf{H})$ values for runs between sets $A$ vs $B$ stays the same after the first 10 epochs. Second, the degree of entanglement is higher between any two epochs when looking at runs across sets $A$ and $B$, while it is weaker when looking at runs within any one of the sets. Finally, test accuracies for set $B$ runs are always higher than those of set $A$ runs, but this trend is not strong for runs within any one set. Note that the minimum loss values are roughly at a similar scale for each dataset and they are all at or below $10^{-2}$.

## B. Computation of $\mathrm{Tr}(\mathbf{H})$

We computed $\mathrm{Tr}(\mathbf{H})$ in our experiments using the Hutchinson's estimator (Hutchinson, 1990),

$$\begin{aligned} Tr(\mathbf{H}) &= Tr(\mathbf{H} \cdot \mathbf{I}) \\ &= Tr(\mathbf{H} \cdot \mathbb{E}[\mathbf{z}\mathbf{z}^T]) \\ &= \mathbb{E}[Tr(\mathbf{H} \cdot \mathbf{z}\mathbf{z}^T)] \\ &= \mathbb{E}[\mathbf{z}^T\mathbf{H} \cdot \mathbf{z}] \\ &\approx \frac{1}{M}\sum_{i=1}^{M} \mathbf{z}_i^T\mathbf{H} \cdot \mathbf{z}_i \\ &= \frac{1}{M}\sum_{i=1}^{M} \mathbf{z}_i^T \frac{\partial}{\partial\theta}\left(\frac{\partial \ell}{\partial \theta^T}\right) \cdot \mathbf{z}_i \\ &= \frac{1}{M}\sum_{i=1}^{M} \mathbf{z}_i^T \frac{\partial}{\partial\theta}\left(\frac{\partial \ell}{\partial \theta}^T \mathbf{z}_i\right), \end{aligned}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{z}$ is a multi-variate standard Gaussian random variable, and $\mathbf{z}_i$'s are i.i.d. instances of $\mathbf{z}$. The larger the value of $M$, the more accurate the approximation is. We used $M = 30$. To make the above computation efficient, note that the gradient $\frac{\partial \ell}{\partial \theta}$ only needs to be computed once and it can be re-used in the summation over the $M$ samples.

## C. Approximations in Fisher Penalty

In this section, we describe in detail the approximations made when computing the Fisher Penalty. Recall that $\mathrm{Tr}(\mathbf{F})$ can be expressed as

$$\mathrm{Tr}(\mathbf{F}) = \mathbb{E}_{x\sim\mathcal{X}, \hat{y}\sim p_\theta(y|\boldsymbol{x})}\left[\|\frac{\partial}{\partial\theta}\ell(\boldsymbol{x}, \hat{y})\|_2^2\right]. \tag{4}$$

In the preliminary experiments, we found that we can use the norm of the expected gradient rather than the expected norm of the gradient, which is a more direct expression of $\mathrm{Tr}(\mathbf{F})$:

$$\begin{aligned} \nabla\mathbb{E}_{x\sim\mathcal{X}, \hat{y}\sim p_\theta(y|\boldsymbol{x})}\left[\left\|\frac{\partial}{\partial\theta}\ell(\boldsymbol{x}, \hat{y})\right\|_2^2\right] &\approx \frac{1}{N}\sum_{n=1}^{N}\frac{1}{M}\sum_{m=1}^{M}\nabla\left\|\frac{\partial}{\partial\theta}\ell(\boldsymbol{x}_n, \hat{y}_{nm})\right\|_2^2 \\ &\geq \nabla\left\|\frac{1}{NM}\sum_{n=1}^{N}\sum_{m=1}^{M}\frac{\partial}{\partial\theta}\ell(\boldsymbol{x}_n, \hat{y}_{nm})\right\|_2^2, \end{aligned}$$

where $N$ and $M$ are the minibatch size and the number of samples from $p_\theta(y|\boldsymbol{x}_n)$, respectively. This greatly improves the computational efficiency. With $N = B$ and $M = 1$, we end up with the following learning objective function:

$$\ell'(\boldsymbol{x}_{1:B}, y_{1:B}; \boldsymbol{\theta}) = \frac{1}{B}\sum_{i=1}^{B}\ell(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) + \alpha\left\|\frac{1}{B}\sum_{i=1}^{B} g(\boldsymbol{x}_i, \hat{y}_i)\right\|^2. \tag{5}$$

We found empirically that $\left\|\frac{1}{B}\sum_{i=1}^{B} g(\boldsymbol{x}_i, \hat{y}_i)\right\|^2$, which we denote by $\mathrm{Tr}(\mathbf{F}_B)$, and $\mathrm{Tr}(\mathbf{F})$ correlate well during training. To demonstrate this, we train SimpleCNN on the CIFAR-10 dataset with 5 different learning rates (from $10^{-3}$ to $10^{-1}$). The outcome is shown in Figure 15. We see that for most of the training, with the exception of the final phase, $\mathrm{Tr}(\mathbf{F}^B)$ and $\mathrm{Tr}(\mathbf{F})$ correlate extremely well. Equally importantly, we find that using a large learning affects both $\mathrm{Tr}(\mathbf{F}_B)$ and $\mathrm{Tr}(\mathbf{F})$, which further suggests the two are closely connected.

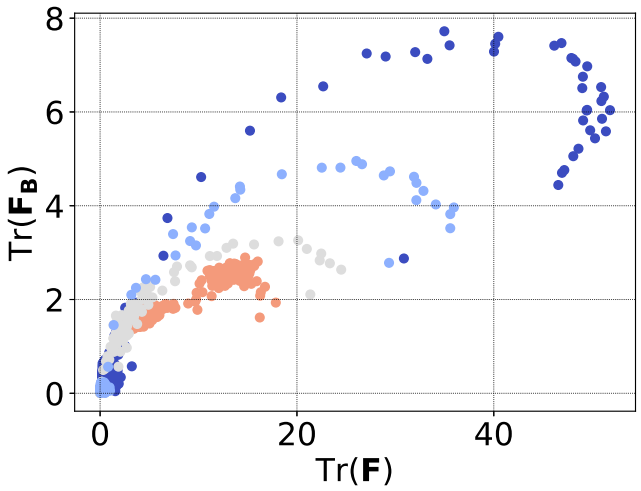
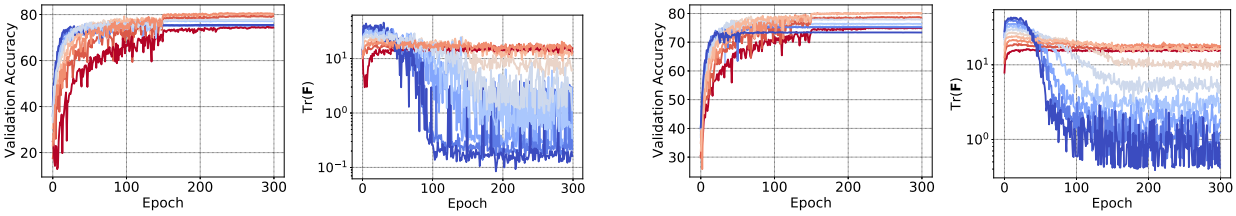



Figure 15: Correlation between $\mathrm{Tr}(\mathbf{F})$ and $\mathrm{Tr}(\mathbf{F}^B)$ for SimpleCNN trained on the CIFAR-10 dataset. Blue to red color denotes learning rates from $10^{-3}$ to $10^{-1}$. The value of $\mathrm{Tr}(\mathbf{F})$ and $\mathrm{Tr}(\mathbf{F}^B)$ correlate strongly for the most of the training trajectory. Using large learning rate reduces both $\mathrm{Tr}(\mathbf{F})$ and $\mathrm{Tr}(\mathbf{F}^B)$.




(a) Fisher Penalty with $f = 10$

(b) Fisher Penalty with $f = 1$

Figure 16: A comparison between the effect of recomputing Fisher Penalty gradient every 10 iterations (left) or every iteration (right), with respect to validation accuracy and $\mathrm{Tr}(\mathbf{F})$. We denote by $f$ the frequency with which we update the gradient. Both experiments result in approximately 80% test accuracy with the best configuration.

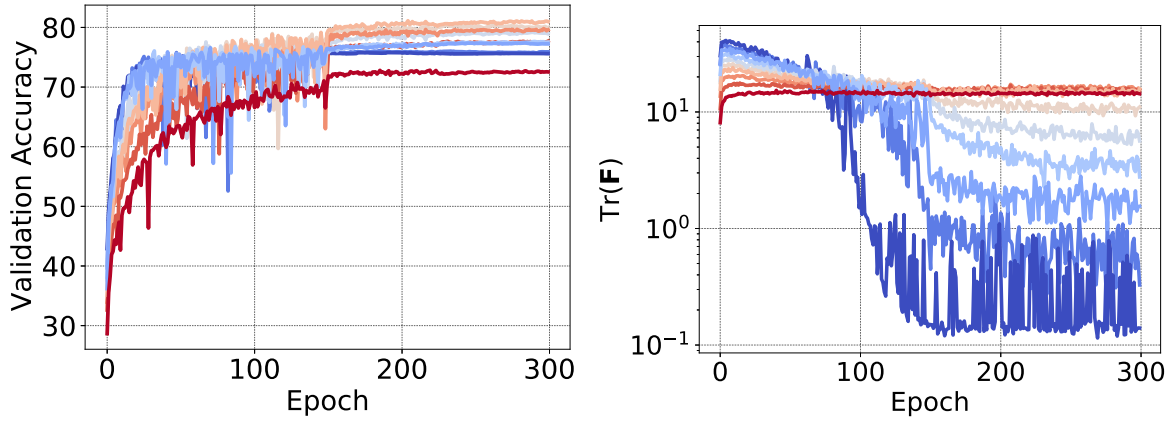



Figure 17: Using Fisher Penalty without the approximation results in a similar generalization performance. We penalize the norm of the gradient rather than norm of the mini-batch gradient (as in Equation 3). We observe that this variant of Fisher Penalty improves generalization to a similar degree as the version of Fisher Penalty used in the paper (c.f. Figure 16.), achieving 79.7% test accuracy.

We also update the gradient of $\mathrm{Tr}(\mathbf{F}^B)$ only every 10 optimization steps. We found empirically it does not affect generalization performance nor the ability to regularize $\mathrm{Tr}(\mathbf{F})$ in our setting. However, we acknowledge that it is plausible that this choice would have to be reconsidered in training with very large learning rates or with larger models.

Figure 16 compares learning curves of training with FP recomputed every optimization step with every 10th optimization step. For each, we tune the hyperparameter $\alpha$, checking 10 values equally spaced between $10^{-2}$ and $10^{0}$ on a logarithmic scale. We observe that for the optimal value of $\alpha$, both validation accuracy and $\mathrm{Tr}(\mathbf{F})$ are similar between the two runs. Both experiments achieve approximately 80% test accuracy.

Finally, to ensure that using the approximation in Equation 3 does not negatively affect how Fisher Penalty improves generalization or reduces the value of $\mathrm{Tr}(\mathbf{F})$, we experiment with a variant of Fisher Penalty without the approximation. Please recall that we always measure $\mathrm{Tr}(\mathbf{F})$ (i.e. we do not use approximations in computing $\mathrm{Tr}(\mathbf{F})$ that is reported in the plots), regardless of what variant of penalty is used in regularizing the training.

Specifically, we augment the loss function with the norm of the gradient computed on the first example in the mini-batch as follows

$$\ell'(\boldsymbol{x}_{1:B}, y_{1:B}; \boldsymbol{\theta}) = \frac{1}{B}\sum_{i=1}^{B} \ell(\boldsymbol{x}_i, y_i; \boldsymbol{\theta}) + \alpha \left\| g(\boldsymbol{x}_1, \hat{y}_1) \right\|^2 . \tag{6}$$

We apply this penalty in each optimization step. We tune the hyperparameter $\alpha$, checking 10 values equally spaced between $10^{-4}$ and $10^{-2}$ on a logarithmic scale.

Figure 17 summarizes the results. We observe that the best value of $\alpha$ yields 79.7% test accuracy, compared to 80.02% test accuracy yielded by the Fisher Penalty. The effect on $\mathrm{Tr}(\mathbf{F})$ is also very similar. We observe that the best run corresponds to a maximum value of $\mathrm{Tr}(\mathbf{F})$ of 24.16, compared to that of 21.38 achieved by Fisher Penalty. These results suggest that the approximation used in this paper's version of the Fisher Penalty only improves the generalization and flattening effects of Fisher Penalty.

## D. A closer look at the surprising effect of learning rate on the loss geometry in the early phase of training

It is intuitive to hypothesize that the catastrophic Fisher explosion (the initial growth of the value of $\mathrm{Tr}(\mathbf{F})$) occurs during training with a large learning rate but is overlooked due to not sufficiently fine-grained computation of $\mathrm{Tr}(\mathbf{F})$. In this section, we show evidence against this hypothesis based on the literature mentioned in the main text. We also run additional experiments in which we compute the value of $\mathrm{Tr}(\mathbf{F})$ at each iteration.

The surprising effect of the learning rate on the geometry of the loss surface (e.g. the value of $\mathrm{Tr}(\mathbf{F})$) was demonstrated in prior works (Jastrzebski et al., 2019; Golatkar et al., 2019; Lewkowycz et al., 2020; Leclerc & Madry, 2020). In particular, Jastrzebski et al. (2020); Lewkowycz et al. (2020) show that training with a large learning rate rapidly escapes regions of high curvature, where curvature is understood as the spectral norm of the Hessian evaluated at the current point of the loss surface. Perhaps the most direct experimental data against this hypothesis can be found in (Cohen et al., 2021) in Figure 1, where training with Gradient Descent finds regions of the loss surface with large curvature for small learning rate rapidly in the early phase of training.

We also run the following experiment to provide further evidence against the hypothesis. We train SimpleCNN on the CIFAR-10 dataset using two different learning rates, while computing the value of $\mathrm{Tr}(\mathbf{F})$ for every mini-batch. We use 128 random samples in each iteration to estimate $\mathrm{Tr}(\mathbf{F})$.

We find that training with a large learning rate never (even for a single optimization step) enters a region where the value of $\mathrm{Tr}(\mathbf{F})$ is as large as what is reached during training with a small learning rate. Figure 18 shows the experimental data.

We also found similar to hold when varying the batch size, see Section E, which further shows that the observed effects cannot be explained by the difference in learning speed incurred by using a small learning rate.

To summarize, both the published evidence of Jastrzebski et al. (2020); Lewkowycz et al. (2020); Cohen et al. (2021), as well as our additional experiments, are inconsistent with the hypothesis that the results in this paper can be explained by differences in training speed between experiments using large and small learning rates.

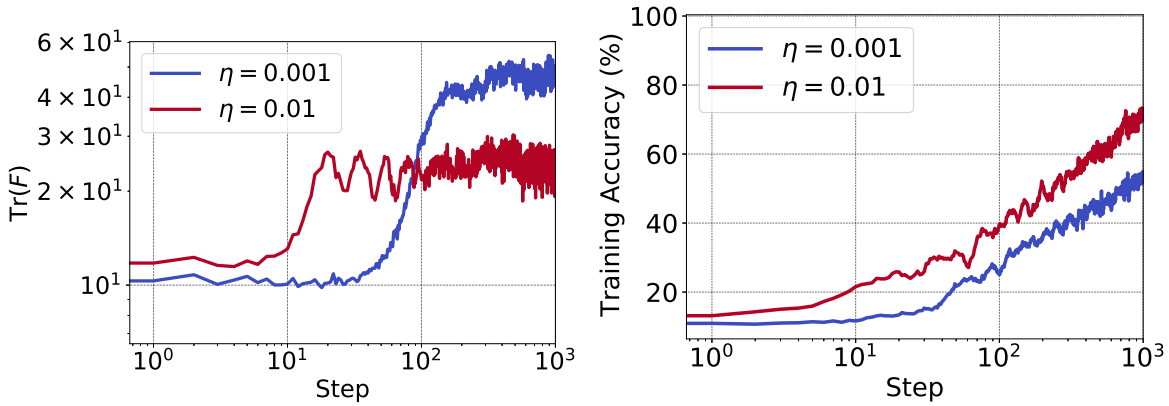



Figure 18: Training with a large learning rate never (even for a single optimization step) enters a region with as large value of $\mathrm{Tr}(\mathbf{F})$ as the maximum value of $\mathrm{Tr}(\mathbf{F})$ reached during training with a small learning rate. We run the experiment using SimpleCNN on the CIFAR-10 dataset with two different learning rates. The left plot shows the value of $\mathrm{Tr}(\mathbf{F})$ computed at each iteration, and the right plot shows training accuracy computed on the current mini-batch (curve has been smoothed for clarity).

## E. Catastrophic Fisher Explosion holds in training with large batch size

In this section, we show preliminary evidence that the conclusions transfer to large batch size training. Namely, we show that (1) catastrophic Fisher explosion also occurs in large batch size training, and (2) Fisher Penalty can improve generalization and close the generalization gap due to using a large batch size (Keskar et al., 2017).

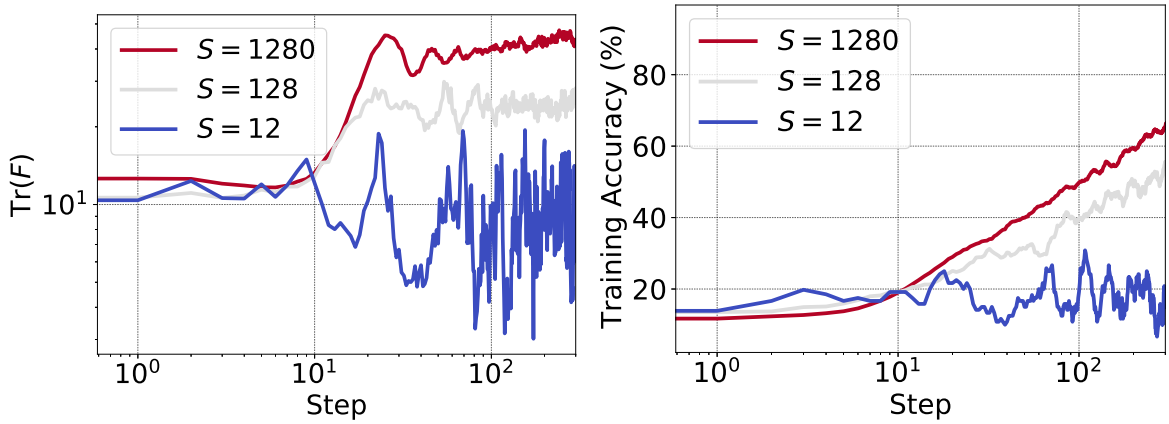



Figure 19: Catastrophic Fisher explosion occurs also in large batch size training. Experiment run on the CIFAR-10 and dataset the SimpleCNN model. The left plot shows the value of $\mathrm{Tr}(\mathbf{F})$ computed at each iteration, and the right plot shows training accuracy computed on the current mini-batch (curve has been smoothed for clarity).

We first train SimpleCNN on the CIFAR-10 dataset using three different batch sizes, while computing the value of $\mathrm{Tr}(\mathbf{F})$ for every mini-batch. We use 128 random samples in each iteration to estimate $\mathrm{Tr}(\mathbf{F})$. Figure 19 summarizes the experiment. We observe that training with a large batch size enters a region of the loss surface with a substantially larger value of $\mathrm{Tr}(\mathbf{F})$ than with the small batch size.

Next, we run a variant of one of the experiments in Table 1. Instead of using a suboptimal (smaller) learning rate, we use a suboptimal (larger) batch size. Specifically, we train SimpleCNN on the CIFAR-10 dataset (without augmentation) with a 10x larger batch size while keeping the learning rate the same. Using a larger batch size results in $3.24\%$ lower test accuracy ($76.94\%$ compared to $73.7\%$ test accuracy, c.f. with Table 1).

We next experiment with Fisher Penalty. We apply the penalty in each optimization step and use the first $128$ examples when computing the penalty. We also use a 2x lower learning rate, which stabilizes training but does not improve generalization on its own (training with this learning rate reaches $73.59\%$ test accuracy). Figure 20 shows $\mathrm{Tr}(\mathbf{F})$ and validation accuracy during training for different values of the penalty. We observe that Fisher Penalty improves test accuracy from $73.59\%$ to $78.7\%$. Applying Fisher Penalty also effectively reduces the peak value of $\mathrm{Tr}(\mathbf{F})$.

Taken together, the results suggest that Catastrophic Fisher explosion holds in large batch size training; using a small batch size improves generalization by a similar mechanism as using a large learning rate, which can be introduced explicitly in the form of Fisher Penalty.

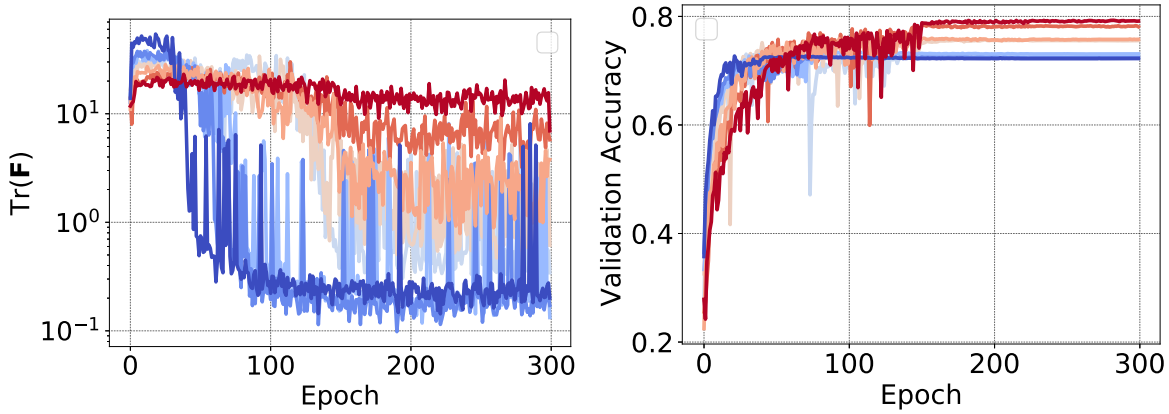
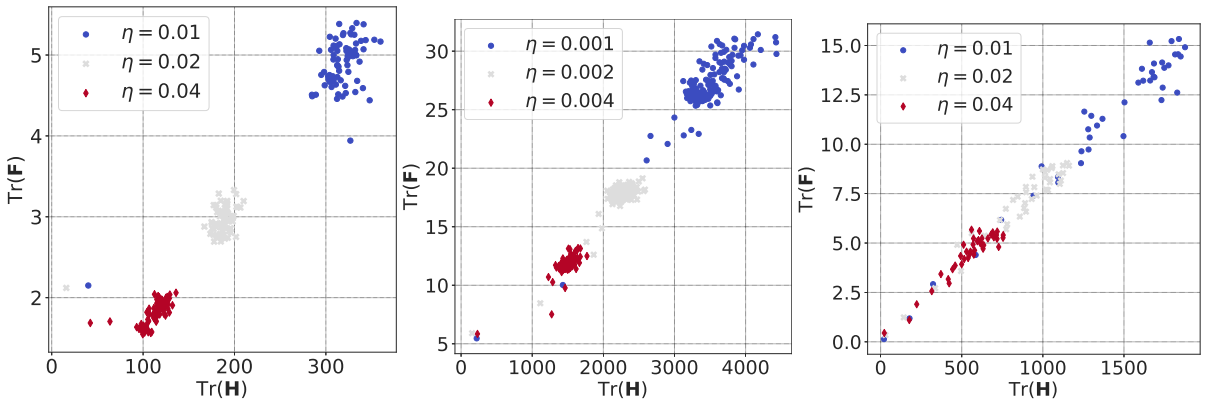



Figure 20: Fisher Penalty improves generalization in large batch size training. Experiment run on the CIFAR-10 dataset (without augmentation) and the SimpleCNN model. Warmer color corresponds to larger coefficient used when applying Fisher Penalty.

## F. $\mathrm{Tr}(\mathbf{H})$ and $\mathrm{Tr}(\mathbf{F})$ correlate strongly

We demonstrate a strong correlation between $\mathrm{Tr}(\mathbf{H})$ and $\mathrm{Tr}(\mathbf{F})$ for DenseNet, ResNet-56 and SimpleCNN in Figure 21. We calculate $\mathrm{Tr}(\mathbf{F})$ using a mini-batch. We see that $\mathrm{Tr}(\mathbf{F})$ has a smaller magnitude (we use a mini-batch gradient) but correlates strongly with $\mathrm{Tr}(\mathbf{H})$.




(a) DenseNet on CIFAR-100 (b) SimpleCNN on CIFAR-10 (c) ResNet-56 on CIFAR-100

Figure 21: Correlation between $\mathrm{Tr}(\mathbf{F})$ and $\mathrm{Tr}(\mathbf{H})$.

## G. Relationship between Fisher Penalty and gradient norm penalty

We give here a short argument that GP might act as a proxy for regularizing $\mathrm{Tr}(\mathbf{F})$. Let $f(\mathbf{x})$ represents the logits of the network, and $\mathcal{L}$ represent the loss. Then $\|\mathbf{g}\|^2 = \|\frac{\partial \mathcal{L}(\mathbf{x},y)}{\partial f(\mathbf{x})}\frac{\partial f(\mathbf{x})}{\partial \theta}\|^2$, and $\mathrm{Tr}(\mathbf{F}) = \|\frac{\partial \mathcal{L}(\mathbf{x},\hat{y})}{\partial f(\mathbf{x})}\frac{\partial f(\mathbf{x})}{\partial \theta}\|^2$. Hence, in particular, reducing the scale of the Jacobian of the logits with respect to weights will reduce both $\|\mathbf{g}\|^2$ and $\mathrm{Tr}(\mathbf{F})$. Empirically, penalizing gradient norm seems to be a less effective regularizer, which suggests that it acts as a proxy for regularizing $\mathrm{Tr}(\mathbf{F})$. A similar argument can be also made for $\mathrm{GP_x}$.

## H. Fisher Penalty Reduces Memorization

We present here a short argument that Fisher Penalty can be seen as reducing the training speed of examples that are both labeled randomly and for which the model makes a random prediction.

Let $\mathcal{D}_R = \{\boldsymbol{x}_i, y_i\}_i$ denote a set of examples where the label $y_i$ is sampled from the discrete uniform distribution $\mathcal{U}$. Let $g(\boldsymbol{x}, y; \boldsymbol{\theta}) = \frac{\partial}{\partial \theta}\ell(\boldsymbol{x}, y; \boldsymbol{\theta})$ denote gradient of the loss function evaluated on an example $(\boldsymbol{x}, y)$.

Assume that the predictive distribution $p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ is uniform for all $\boldsymbol{x} \in \mathcal{D}_R$. Then the expected mean squared norm of gradient of examples in $\mathcal{D}_R$ over different sampling of labels is $\mathbb{E}[\|g(\boldsymbol{x}, y)\|_2^2] = \frac{1}{|\mathcal{D}_R|}\sum_i \mathbb{E}_{y\sim\mathcal{U}}[[\|g(\boldsymbol{x_i}, y)[\|^2] = \frac{1}{|\mathcal{D}_R|}\sum_i \mathbb{E}_{\hat{y}\sim p_{\boldsymbol{\theta}}(\hat{y}|\boldsymbol{x})}[\|g(\boldsymbol{x_i}, \hat{y})_2\|^2]$.

Fisher Penalty aims to penalize the trace of the Fisher Information Matrix. For examples in $\mathcal{D}_R$, $\mathrm{Tr}(\mathbf{F})$ evaluates to $\mathrm{Tr}(\mathbf{F}_R) = \frac{1}{|\mathcal{D}_R|}\mathbb{E}_{\hat{y}\sim p_\theta(y|\boldsymbol{x})}\left[\|g(\boldsymbol{x_i}, \hat{y})\|_2^2\right]$, which under our assumptions is equal to $\mathbb{E}[\|g(\boldsymbol{x}, y)\|^2]$.

We are interested in understanding how Fisher Penalty affects the learning speed of noisy examples. The reduction in the training loss for a given example can be related to its gradient norm using Taylor expansion. Consider the difference in training loss $\Delta\ell = \ell\left(\boldsymbol{x}, y; \boldsymbol{\theta} - \eta g(\boldsymbol{x}, y)\right) - \ell\left(\boldsymbol{x}, y; \boldsymbol{\theta}\right)$ after performing a single step of gradient descent on this example. Using first-order Taylor expansion we arrive at $\Delta\ell \approx -\eta\|g(\boldsymbol{x}, y)\|_2^2$.

Taken together, penalizing $\|g(\boldsymbol{x}, y)\|_2$, which is achieved by penalizing $\mathrm{Tr}(\mathbf{F})_R$, can be seen as slowing down learning on noisy examples.

However, in practice, we apply Fisher Penalty to all examples in the training set because we do not know which ones are corrupted. Consider $\mathcal{D} = \mathcal{D}_R \cup \mathcal{D}_C$, where $\mathcal{D}$ is the whole training set and $\mathcal{D}_C$ denotes the subset with clean (not altered) labels. Then, $\mathrm{Tr}(\mathbf{F}) = \mathrm{Tr}(\mathbf{F}_R) + \mathrm{Tr}(\mathbf{F}_C)$, where $\mathrm{Tr}(\mathbf{F})$ denotes trace of the FIM evaluated on the whole dataset, and $\mathrm{Tr}(\mathbf{F}_C)$ ($\mathrm{Tr}(\mathbf{F}_R)$) denotes the trace of the FIM on the clean (noisy) subset of the dataset.

Hence, if $\mathrm{Tr}(\mathbf{F}_R) \gg \mathrm{Tr}(\mathbf{F}_C)$, we can expect Fisher Penalty to disproportionately slow down training of noisy examples. This assumption is likely to hold because the clean examples tend to be learned much earlier in training than noisy ones (Arpit et al., 2017). In experiments, we indeed observe that the gradient norm of examples with noisy labels is disproportionately affected by Fisher Penalty, and also that learning on noisy examples is slower.

## I. Additional Experimental Details

### I.1. Early phase $\mathrm{Tr}(\mathbf{F})$ correlates with final generalization

Here, we describe additional details for experiments in Section 3.

In the experiments with batch size, for CIFAR-10, we use batch sizes 100, 500 and 700, and $\epsilon = 1.2$. For CIFAR-100, we use batch sizes 100, 300 and 700, and $\epsilon = 3.5$. These thresholds are crossed between 2 and 7 epochs across different hyperparameter settings. The remaining details for CIFAR-100 and CIFAR-10 are the same as described in the main text. The optimization details for these datasets are as follows.

**ImageNet**: No data augmentation was used in order to allow training loss to converge to small values. We use a batch size of 256. Training is done using SGD with momentum set to 0.9, weight decay set to $1e-4$, and with base learning rates in $\{0.001, 0.01, 0.1\}$. Learning rate is dropped by a factor of 0.1 after 29 epochs and training is ended at around 50 epochs at which most runs converge to small loss values. No batch normalization is used and weight are initialized using Fixup (Zhang et al., 2019). For each hyperparameter setting, we run two experiments with different random seeds (due to the computational overhead). We compute $\mathrm{Tr}(\mathbf{F})$ using 2500 samples (similarly to (Jastrzebski et al., 2020)).

**CIFAR-10**: We used random flipping as data augmentation. In the experiments with variation in learning rates (used $\{0.007, 0.01, 0.05\}$), we use a batch size of 256. In the experiments with variation in batch size (used 100, 500 and 700), we use a learning rate of 0.02. Training is done using SGD with momentum set to 0.9, weight decay set to $1e-5$. Learning rate is dropped by a factor of 0.5 at epochs 60, 120, and 170, and training is ended at 200 epochs at which most runs converge to small loss values. No batch normalization is used and weight are initialized using (Arpit et al., 2019). For each hyperparameter setting, we run 32 experiments with different random seeds. We compute $\mathrm{Tr}(\mathbf{F})$ using 5000 samples.

**CIFAR-100**: No data augmentation was used for CIFAR-100 to allow training loss to converge to small values. We used random flipping as data augmentation for CIFAR-10. In the experiments with variation in learning rates (used $\{0.005, 0.001, 0.01\}$), we use a batch size of 100. In the experiments with variation in batch size (used 100, 300 and 700), we use a learning rates of 0.02. Training is done using SGD with momentum set to 0.9, weight decay set to $1e-5$. Learning rate is dropped by a factor of 0.5 at epochs 60, 120, and 170, and training is ended at 200 epochs at which most runs converge to small loss values. No batch normalization is used and the weights are initialized using (Arpit et al., 2019). For each hyperparameter setting, we run 32 experiments with different random seeds. We compute $\mathrm{Tr}(\mathbf{F})$ using 5000 samples.

### I.2. Fisher Penalty

Here, we describe the remaining details for the experiments in Section 4. We first describe how we tune hyperparameters in these experiments. In the remainder of this section, we describe each setting used in detail.

**Tuning hyperparameters** In all experiments, we refer to the optimal learning rate $\eta^*$ as the learning rate found using grid search. In most experiments, we check 5 different learning rate values uniformly spaced on a logarithmic scale, usually between $10^{-2}$ and $10^{0}$. In some experiments, we adapt the range to ensure that it includes the optimal learning rate. We tune the learning rate only once for each configuration (i.e. we do not repeat it for different random seeds).

In the first setting, for most experiments involving gradient norm regularizers, we use $10\times$ smaller learning rate than $\eta^*$. For TinyImageNet, we use $30\times$ smaller learning rate than $\eta^*$. To pick the regularization coefficient $\alpha$, we evaluate 10 different values uniformly spaced on a logarithmic scale between $10^{-1} \times v$ to $10^{1} \times v$ with $v \in \mathbb{R}_+$. We choose the best performing $\alpha$ according to validation accuracy. We pick the value of $v$ manually with the aim that the optimal $\alpha$ is included in this range. We generally found that $v = 0.01$ works well for GP, $\mathrm{GP}_\mathrm{r}$, and FP. For $\mathrm{GP}_\mathrm{x}$ we found in some experiments that it is necessary to pick larger values of $v$.

**Measuring** $\mathrm{Tr}(\mathbf{F})$ We measure $\mathrm{Tr}(\mathbf{F})$ using the number of examples equal to the batch size used in training. For experiments with Batch Normalization layers, we use Batch Normalization in evaluation mode due to the practical reason that computing $\mathrm{Tr}(\mathbf{F})$ uses a batch size of 1, and hence $\mathrm{Tr}(\mathbf{F})$ is not defined for a network with Batch Normalization layers in training mode.

**DenseNet on the CIFAR-100 dataset** We use the DenseNet (L=40, k=12) configuration from (Huang et al., 2017). We largely follow the experimental setting in (Huang et al., 2017). We use the standard data augmentation (where noted) and data normalization for CIFAR-100. We hold out random 5000 examples as the validation set. We train the model using SGD with a momentum of 0.9, a batch size of 128, and a weight decay of 0.0001. Following (Huang et al., 2017), we train for 300 epochs and decay the learning rate by a factor of 0.1 after epochs 150 and 225. To reduce variance, in testing we update Batch Normalization statistics using 100 batches from the training set.

**Wide ResNet on the CIFAR-100 dataset** We train Wide ResNet (depth 44 and width 3, without Batch Normalization layers). We largely follow experimental setting in (He et al., 2016).We use the standard data augmentation and data normalization for CIFAR-100. We hold out random 5000 examples as the validation set. We train the model using SGD with a momentum of 0.9, a batch size of 128, and a weight decay of 0.0010. Following (He et al., 2016), we train for 300 epochs and decay the learning rate by a factor of 0.1 after epochs 150 and 225. We remove Batch Normalization layers. To ensure stable training we use the SkipInit initialization (De & Smith, 2020).

**VGG-11 on the CIFAR-100 dataset** We adapt the VGG-11 model (Simonyan & Zisserman, 2015) to CIFAR-100. We do not use dropout nor Batch Normalization layers. We hold out random 5000 examples as the validation set. We use the standard data augmentation (where noted) and data normalization for CIFAR-100. We train the model using SGD with a momentum of 0.9, a batch size of 128, and a weight decay of 0.0001. We train the model for 300 epochs and decay the

learning rate by a factor of 0.1 after every 40 epochs starting from epoch 80.

**SimpleCNN on the CIFAR-10 dataset** We also run experiments on the CNN example architecture from the Keras example repository (Chollet & others, 2015)[2], which we change slightly. Specifically, we remove dropout and reduce the size of the final fully-connected layer to 128. We train it for 300 epochs and decay the learning rate by a factor of 0.1 after the epochs 150 and 225. We train the model using SGD with a momentum of 0.9, and a batch size of 128.

**Wide ResNet on the TinyImageNet dataset** We train Wide ResNet (depth 44 and width 3, with Batch Normalization layers) on TinyImageNet Le & Yang (2015). TinyImageNet consists of a subset of 100,000 examples from ImageNet that we downsized to 32×32 pixels. We train the model using SGD with a momentum of 0.9, a batch size of 128, and a weight decay of 0.0001. We train for 300 epochs and decay the learning rate by a factor of 0.1 after epochs 150 and 225. We do not use validation in TinyImageNet due to its larger size. To reduce variance, in testing we update Batch Normalization statistics using 100 batches from the training set.

### I.3. Fisher Penalty Reduces Memorization

Here, we describe additional experimental details for Section 4.1. We use two configurations described in Section I.2: VGG-11 trained on CIFAR-100 dataset, and Wide ResNet trained on the CIFAR-100 dataset. We tune the regularization coefficient $\alpha$ in the range $\{0.01, 0.1, 0.31, 10\}$, with the exception of $\text{GP}_\text{x}$ for which we use the range $\{10, 30, 100, 300, 1000\}$. We tuned the mixup coefficient in the range $\{0.4, 0.8, 1.6, 3.2, 6.4\}$. We removed weight decay in these experiments. We use validation set for early stopping, as commonly done in the literature.

### I.4. Early $\text{Tr}(\mathbf{F})$ influences final curvature

**CIFAR-10**: We used random flipping as data augmentation for CIFAR-10. We use a learning rate of 0.02 for all experiments. Training is done using SGD with momentum 0.9, weight decay $1e-5$, and batch size as shown in figures. The learning rate is dropped by a factor of 0.5 at 80, 150, and 200 epochs, and training is ended at 250 epochs. No batch normalization is used and the weights are initialized using (Arpit et al., 2019). For each batch size, we run 32 experiments with different random seeds. We compute $\text{Tr}(\mathbf{F})$ using 5000 samples.

**CIFAR-100**: No data augmentation is used. We use a batch size of 100 for all experiments. Training is done using SGD with momentum 0.9, weight decay $1e-5$, and with base learning rate as shown in figures. The learning rate is dropped by a factor of 0.5 at 80, 150, and 200 epochs, and training is ended at 250 epochs. No batch normalization is used and the weights are initialized using (Arpit et al., 2019). For each learning rate, we run 32 experiments with different random seeds. We compute $\text{Tr}(\mathbf{F})$ using 5000 samples.

[2] Accessible at https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py.