RECYCLING MODEL UPDATES IN FEDERATED LEARNING: ARE GRADIENT SUBSPACES LOW-RANK?

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we question the rationale behind propagating large numbers of parameters through a distributed system during federated learning. We start by examining the rank characteristics of the subspace spanned by gradients (i.e., the gradient-space) in centralized model training, and observe that the gradient-space often consists of a few leading principal components accounting for an overwhelming majority ($95 - 99\%$) of the explained variance. Motivated by this, we propose the "Look-back Gradient Multiplier" (`LBGM`) algorithm, which utilizes this low-rank property of the gradient-space in federated learning. Operationally, `LBGM` recycles the gradients between model update rounds to significantly reduce the number of parameters to be propagated through the system. We analytically characterize the convergence behavior of `LBGM`, revealing the nature of the trade-off between communication savings and model performance. Our subsequent experimental results demonstrate the improvement `LBGM` obtains on communication overhead compared to several federated learning baselines. Additionally, we show that `LBGM` is a general plug-and-play algorithm that can be used standalone or stacked on top of existing sparsification techniques for distributed model training.

1 INTRODUCTION

Federated Learning (FL) (Konečnỳ et al., 2016) has emerged as a popular paradigm for distributing machine learning (ML) across a set of computing devices. By having each device conduct local model updates, FL substitutes raw data transmissions with model parameter transmissions, promoting data privacy (Shokri & Shmatikov, 2015; Azam et al., 2021) and reducing communication requirements (Wang et al., 2020a; Hosseinalipour et al., 2020). At the same time, overparameterized neural networks (NN) are becoming ubiquitous in the ML models trained by FL, e.g., in computer vision (Ren et al., 2015; He et al., 2016; 2017; Liu et al., 2016; Huang et al., 2017) and natural language processing (Brown et al., 2020; Vaswani et al., 2017; Devlin et al., 2019; Radford et al., 2019; Liu et al., 2019a), which reintroduces communication burden. While NNs have parameters in the range of a few million (AlexNet (Krizhevsky et al., 2012), VGG (Simonyan & Zisserman, 2014), ResNet (He et al., 2016)) to several billions (GPT-3 (Brown et al., 2020), GPT-2 (Radford et al., 2019), Turing NLG (tur, 2020)), prior works have demonstrated that a majority of these parameters are often irrelevant (Frankle & Carbin, 2019; Liu et al., 2019b; Han et al., 2015; Li et al., 2017; LeCun et al., 1990) in optimization and inference. This presents an opportunity to reduce communication in FL by transmitting lighter representations of the model, conventionally achieved through compression/sparsification (Wang et al., 2018; Seide et al., 2014; Alistarh et al., 2017; Vogels et al., 2019).

In this work, we take a fresh look at the "overparameterization" of NN training, through the lens of rank characteristics of the subspaces spanned by gradients (i.e., gradient-space) generated during (stochastic) gradient descent (SGD). Our first fundamental question is: *Can we observe the effect of overparameterization in gradient descent-based optimization of NNs?* And if so: *Can this property be used to reduce communication overhead in FL?* Our main hypothesis is that

"the subspace spanned by the gradients generated during SGD is low-rank". **(H1)**

This leads us to propose a technique that instead of propagating a million/billion-dimensional vector (i.e., gradient) over the system per iteration of FL, only relies on propagating a single scalar in the majority of the iterations. Our algorithm introduces a new class of techniques to the literature which

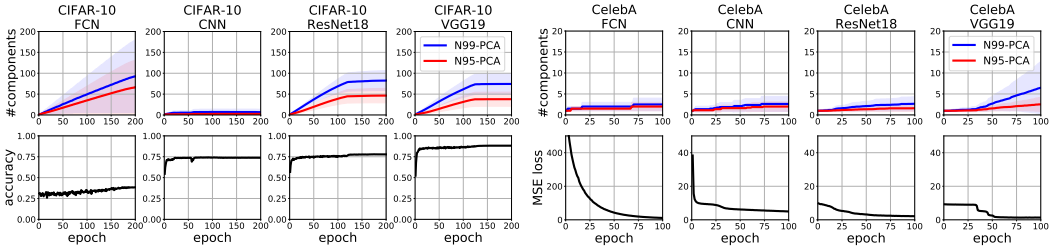

Figure 1: *PCA Components Progression.* Top row shows the number of components that account for 99% (N99-PCA in blue) and 95% (N95-PCA in red) explained variance of all the gradients generated during gradient descent iterations. Bottom row shows the performance of the model on the test data. The results are presented for: (i) CIFAR-10 classification (left 4 columns), and (ii) CelebA regression task (right 4 columns).

are built on reusing/recycling the previous gradients to represent the newly generated gradients at the devices in FL. The main contributions of this work can be summarized as follows:

- We give a fresh perspective on the effect of overparameterization on the optimization of NNs by analyzing gradient-based model training through the lens of rank characteristics of the gradient-space. We show that the principal components of the gradient-space (principal gradients) can be approximated in terms of actual gradients generated during the model training process.
- Our insights lead us to develop the "Look-back Gradient Multiplier" (`LBGM`) algorithm to significantly reduce the communication overhead in FL. `LBGM` recycles the previously transmitted gradients to represent the newly-generated gradients at each device with a single scalar. We further analytically investigate the convergence characteristics of this algorithm.
- Our experiments show the communication savings obtained via `LBGM` in FL both as a standalone solution and a plug-and-play method used with other gradient compression techniques, e.g., top-K. We further reveal that `LBGM` can be extended to distributed training, e.g., `LBGM` with SignSGD (Bernstein et al., 2018) substantially reduces communication overhead in multi-GPU systems.

2 A Gradient-Space Odyssey

We first experimentally study the effect of overparameterization on the gradient-space spanned by gradients generated during NN optimization. Given a centralized ML training task (e.g., classification, regression, segmentation), we exploit principal component analysis (PCA) (Pearson, 1901) to answer the following: *How many principal components explain the* 99% *and* 95% *variance (termed* N99-PCA *and* N95-PCA*, respectively; together* N-PCA*) of all the gradients generated during model training?*

We start with 4 different NN architectures: (i) fully-connected neural network (FCN), (ii) convolutional neural network (CNN), (iii) ResNet18 (He et al., 2016), and (iv) VGG19 (Simonyan & Zisserman, 2014); trained on 2 different datasets: CIFAR-10 (Krizhevsky et al., 2009) and CelebA (Liu et al., 2015), which have classification and regression (face landmarks point prediction) learning tasks, respectively. We then compute the N-PCA for each epoch by applying PCA on the set of gradients accumulated until that epoch to get the N-PCA progression as shown in Fig. 1. Refer to Algorithm 2 in **Appendix C** for the pseudocode. The results depicted in Fig. 1 agree with our hypothesis (**H1**): both N99-PCA and N95-PCA are significantly lower than that the total number of gradients calculated during model training irrespective of model/dataset/learning task, e.g., Fig. 1 shows that the number of principal gradients (red and blue lines in top plots) are substantially lower (often as low as 10% of number of epochs, i.e., gradients generated) for both datasets. We further find that (**H1**) holds for additional experiments using several datasets: CIFAR-100 (Krizhevsky et al., 2009), MNIST (LeCun & Cortes, 2010), FMNIST (Xiao et al., 2017), CelebA (Liu et al., 2015), PascalVOC (Everingham et al., 2010), COCO (Lin et al., 2014); models: U-Net (Ronneberger et al., 2015), SVM (Cortes & Vapnik, 1995); and tasks: semantic segmentation and multi-regression in **Appendix D.1**. It is also worth noting (especially on CIFAR-10) that variations in N-PCA across models are not necessarily related to the model performance (CNN performs almost as well as ResNet18 but has much lower N-PCA; Fig. 1 – columns 2 & 3) or complexity (CNN has more parameters than FCN but has lower N-PCA; Fig. 1 – columns 1 & 2). The fact that rank deficiency of the gradient-space is not a consequence of model complexity or model performance suggests that the gradient-space of state-of-the-art large-scale ML models could be represented using a few principal gradient directions.

N-PCA and FL. In an "ideal" FL (McMahan et al., 2017) framework, if both the server and the workers/devices have the principal gradients (principal components of the gradient-space), then

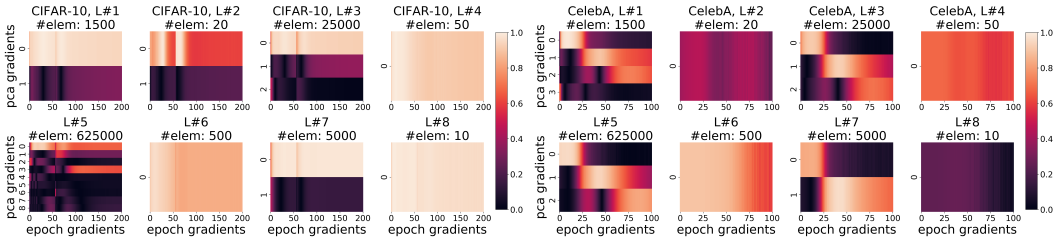

Figure 2: *Overlap of Actual and Principal Gradient*. The heatmap shows the pairwise cosine similarity of actual (epoch) gradients and principal (pca) gradient. Epoch gradients have a substantial overlap with one or more pca gradients and consecutive epoch gradients show only a gradual shift w.r.t. pca gradients. The results are shown for a CNN classifier trained on CIFAR-10 (left 4 columns) and CelebA (right 4 columns) datasets. Each subplot is marked with #L, the layer number of the CNN and #elem, the number of elements in each layer.

the newly generated gradients can be transmitted by sharing the principal gradients' multipliers (PGM), i.e., projections of the generated gradients on the principal gradients. PGMs and the principal gradients can together be used to reconstruct the generated gradients of the devices at the server. This would dramatically reduce the communication cost by sharing scalar PGMs instead of the gradient vector. However, this setting is impractical since: (i) it is infeasible to obtain the principal gradients prior to the training, and (ii) calculating PCA/SVD is computationally intense. We thus look for an efficient online approximation of the principal gradients. In practice, a noisy approximation of principal gradients is a feasible solution since SGD is resilient to noisy updates (Friedlander & Schmidt, 2012), and often benefits from them in terms of generalization error (Wu et al., 2020).

Overlap of Actual Gradients and N-PCA. To approximate principal gradients, we exploit an observation made in Fig. 1 (and further in Appendix D.1): N-PCA mostly remains constant over time suggesting that the gradients change gradually during SGD iterations. To further investigate this, in Fig. 2 (and further in Appendix D.2) we plot the cosine similarity between principal gradients and actual gradients as a heatmap[‡]. We observe that (i) the cosine similarity of actual gradients with principal gradients varies gradually over time, and (ii) actual gradients have a high cosine similarity with one or more of the principal gradients. These observations lead to our second hypothesis:

"Principal gradients can be approximated using a subset of the actual gradients". **(H2)**

"Look-back Gradients" as Principal Gradients Approximation. Our above-mentioned observations suggest that there should be a significant overlap among consecutive gradients generated during SGD iterations. This is further verified in Fig. 3 (and in Appendix D.3), where we plot the pairwise cosine similarity of consecutive gradients generated during SGD iterations[‡]. For example, consider the boxes marked B1, B2, ... in layer 1 (L#1 in Fig. 3) for reference. Any of the gradients in these boxes can be used to approximate other gradients within the box. Also, interestingly, the number of such boxes that can be drawn is correlated with the corresponding number of principal gradients. These observations substantiate (**H2**). Based on (**H2**), we next propose our *Look-back Gradient Multiplier* (LBGM) algorithm that utilizes a subset of actual gradients, termed "look-back gradients", in order to reuse/recycle gradients to be transmitted in a federated learning (FL) framework.

3 Look-back Gradient Multiplier

Federated Learning (FL) considers a system of K workers/devices indexed $1, \cdots, K$ as shown in Fig. 4. Each worker k possesses a local dataset $\mathcal{D}_k$. The goal of the system is to minimize the global loss function $F(\cdot)$ expressed through the following optimization problem:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^M} \left\{ F(\boldsymbol{\theta}) \triangleq \sum_{k=1}^{K} \omega_k F_k(\boldsymbol{\theta}) \right\}, \tag{1}$$

where M is the dimension of the model $\boldsymbol{\theta}$, $\omega_k = n_k/N$ ($n_k = |\mathcal{D}_k|$ and $N = \sum_{k=1}^{K} n_k$), and $F_k(\boldsymbol{\theta}) = \sum_{d \in \mathcal{D}_k} f_k(\boldsymbol{\theta}; d)/n_k$ is the local loss at worker k, with $f_k(\boldsymbol{\theta}; d)$ denoting the loss function over data sample d given parameter $\boldsymbol{\theta}$. FL tackles (1) via engaging the workers in local model training on their own datasets, often using SGD. The local models are periodically transferred to and aggregated at the main server after τ local updates, forming a global model that is used to synchronize

[‡]Refer to Algorithm 2 in Appendix C for the detailed pseudocode.

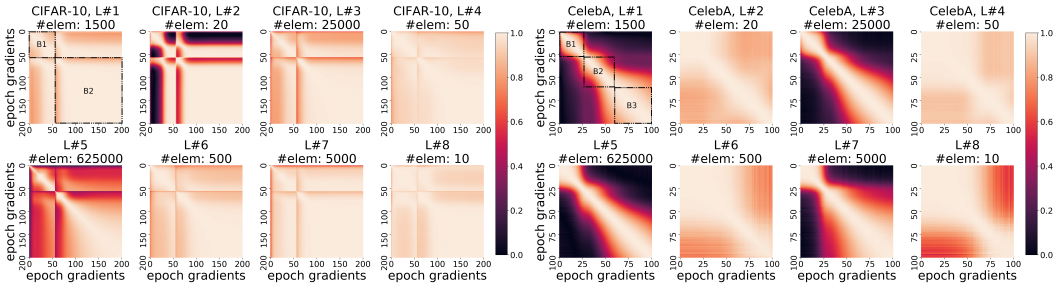

Figure 3: *Similarity among Consecutive Generated Gradients*. The overlap of consecutive gradients in terms of cosine similarity reveals that there is a gradual change in directions of gradients over epochs. This similarity can thus be used to represent the newly generated gradients with low approximation error in terms of the previously generated gradients. In particular, this suggests that the gradients generated during SGD should not be discarded instead reused/recycled to represent the gradients generated in the next epochs.

the workers before starting the next round of local model training. During each global aggregation round t, worker k updates its parameters $\boldsymbol{\theta}_k^{(t,b)}$ as: $\boldsymbol{\theta}_k^{(t,b+1)} \leftarrow \boldsymbol{\theta}_k^{(t,b)} - \eta \boldsymbol{g}_k(\boldsymbol{\theta}_k^{(t,b)})$, where $\boldsymbol{g}_k(\boldsymbol{\theta}_k^{(t,b)})$ is the stochastic gradient at local step b, and η is the step size. During a vanilla FL aggregation, the global model parameters are updated as $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \sum_{k=1}^{K} \omega_k \boldsymbol{g}_k^{(t)}$, where $\boldsymbol{g}_k^{(t)} = \sum_{b=0}^{\tau-1} \boldsymbol{g}_k(\boldsymbol{\theta}_k^{(t,b)})$ is accumulated stochastic gradient (ASG) at worker k. We define $\nabla F_k(\boldsymbol{\theta}_k^{(t)}) = \sum_{b=0}^{\tau-1} \nabla F_k(\boldsymbol{\theta}_k^{(t,b)})$ and $\boldsymbol{d}_k^{(t)} = \boldsymbol{g}_k^{(t)}/\tau$ as the corresponding accumulated true gradient and normalized ASG respectively.

Indexing and Notations. In *superscripts with parenthesis* expressed as tuples, the first element denotes the global aggregation round while the second element denotes the local update round, e.g., $\boldsymbol{g}_k^{(t,b)}$ is the gradient at worker k at global aggregation round t at local update b. *Superscripts without parenthesis* denote the index for look-back gradients, e.g., ℓ in $\rho_k^{(t),\ell}$ defined below.

`LBGM` **Algorithm.** `LBGM` consists of three main steps: (i) workers initialize and propagate their look-back gradients (LBGs) to the server; (ii) workers estimate look-back coefficients (LBCs), which is the scalar projection of subsequent aggregated gradients on their LBGs, and the look-back phase (LBP), which is the angle between the accumulated gradient and the LBG; (iii) workers update their LBGs and propagate them to the server if the value of LBP passes a threshold, otherwise they only transmit the scalar LBC. Thus, `LBGM` propagates only a subset of actual gradients generated at the devices to the server. The intermediate global aggregation steps between two LBG propagation rounds only involve transfer of a *single scalar*, i.e., LBC, from each worker, instead of the entire vector of accumulated gradient. In `LBGM` (see Fig. 4), the local model training is conducted the same as vanilla FL. However, the model aggregations at the server are conducted via the following rule:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \sum_{k=1}^{K} \omega_k \widetilde{\boldsymbol{g}}_k^{(t)}, \tag{2}$$

where $\widetilde{\boldsymbol{g}}_k^{(t)}$ is the approximation of worker k's accumulated stochastic gradient given by Definition 1.

Definition 1. ***(Gradient Approximation in*** `LBGM`***)*** *Given the accumulated stochastic gradient $\boldsymbol{g}_k^{(t)}$, and the LBG $\boldsymbol{g}_k^{\ell}$, the gradient approximation $\widetilde{\boldsymbol{g}}_k^{(t)}$ recovered by the server is given by:*

$$\widetilde{\boldsymbol{g}}_k^{(t)} = \rho_k^{(t),\ell} \boldsymbol{g}_k^{\ell} \text{ and } \left\| \rho_k^{(t),\ell} \boldsymbol{d}_k^{\ell} \right\| = \left\| \boldsymbol{d}_k^{(t)} \cos(\alpha_k^{(t),\ell}) \right\|, \tag{D1}$$

where LBC $\rho_k^{(t),\ell} = \langle \boldsymbol{g}_k^{(t)}, \boldsymbol{g}_k^{\ell} \rangle / \|\boldsymbol{g}_k^{\ell}\|^2$ is the projection of the accumulated gradient $\boldsymbol{g}_k^{(t)}$ on the LBG $\boldsymbol{g}_k^{\ell}$, and $\alpha_k^{(t),\ell}$ denotes the angle between $\boldsymbol{g}_k^{(t)}$ and $\boldsymbol{g}_k^{\ell}$ (see Fig. 4(a)).

`LBGM` initializes the LBGs $\boldsymbol{g}_k^{\ell}$ with the first actual gradients propagated by the devices at $t = 1$. For the subsequent aggregations, each worker k shares only its LBC $\rho_k^{(t),\ell}$ with the server if the value of LBP error $\sin^2(\alpha_k^{(t),\ell})$ is below a threshold, i.e., $\sin^2(\alpha_k^{(t),\ell}) \leq \delta_k^{\text{threshold}}$, where $\delta_k^{\text{threshold}} \in [0, 1]$ is a tunable parameter; otherwise it updates the LBG via transmitting the entire gradient to the server. The details of `LBGM` are given in Algorithm 1. We next conduct convergence analysis for `LBGM`.

Assumptions: It is presumed that the local loss functions are bounded below: $\min_{\boldsymbol{\theta} \in \mathbb{R}^M} F_k(\boldsymbol{\theta}) > -\infty, \forall k$. Let $\|\cdot\|$ denote the 2-norm. In the analysis we use the following assumptions (Wang et al., 2020a; Friedlander & Schmidt, 2012; Hosseinalipour et al., 2020; Li et al.; Stich, 2019):

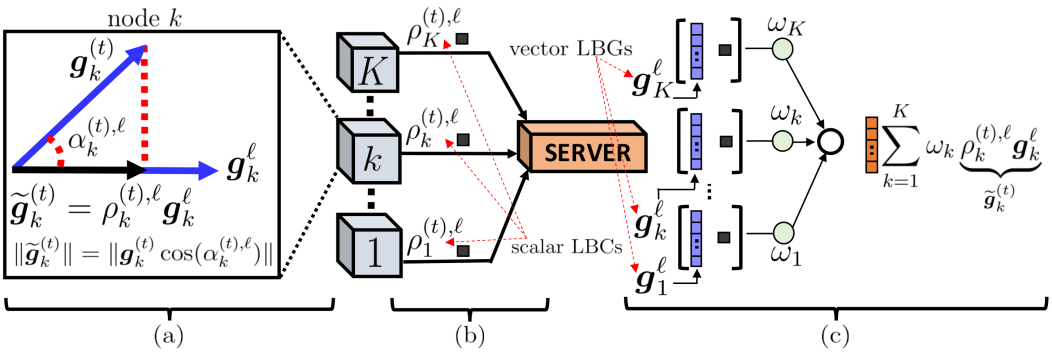

Figure 4: *Look-back Gradient Multiplier*. (a) The Look-back Coefficients (LBCs) are the projection of accumulated stochastic gradient at the workers on their Look-back Gradients (LBGs). (b) Scalar LBCs, i.e., the $\rho_k^{(t),\ell}$, are transmitted to the server. (c) LBG-based gradient approximations are reconstructed at the server.

Algorithm 1 `LBGM`: Look-back Gradient Multiplier

Training at worker k:

1: Update local parameters: $\boldsymbol{\theta}_k^{(t,0)} \leftarrow \boldsymbol{\theta}^{(t)}$, and initialize gradient accumulator: $\boldsymbol{g}_k^{(t)} \leftarrow \mathbf{0}$.
2: **for** $b = 0$ to $(\tau\text{-}1)$ **do**
3: Sample a minibatch of datapoints $\mathcal{B}_k$ from $\mathcal{D}_k$ and compute $\boldsymbol{g}_k^{(t,b)} = \sum_{d\in\mathcal{B}_k} \nabla f_k(\boldsymbol{\theta}_k^{(t,b)}; d)/|\mathcal{B}_k|$.
4: Update local parameters: $\boldsymbol{\theta}_k^{(t,b+1)} \leftarrow \boldsymbol{\theta}_k^{(t,b)} - \eta \cdot \boldsymbol{g}_k^{(t,b)}$, and accumulate gradient: $\boldsymbol{g}_k^{(t)} \leftarrow \boldsymbol{g}_k^{(t)} + \boldsymbol{g}_k^{(t,b)}$.
5: **end for**
6: Calculate the look-back phase (LBP) error: $\sin^2(\alpha_k^{(t),\ell}) = 1 - \left(\langle \boldsymbol{g}_k^{(t)}, \boldsymbol{g}_k^{\ell}\rangle / \left(\|\boldsymbol{g}_k^{(t)}\| \times \|\boldsymbol{g}_k^{\ell}\|\right)\right)^2$
7: **if** $\sin^2(\alpha_k^{(t),\ell}) \leq \delta_k^{\text{threshold}}$ **then** ▷ checking the LBP error
8: Send scalar look-back coefficient (LBC) to the server: $\boldsymbol{\mu}_k^{(t)} \leftarrow \rho_k^{(t),\ell} = \langle \boldsymbol{g}_k^{(t)}, \boldsymbol{g}_k^{\ell}\rangle / \|\boldsymbol{g}_k^{\ell}\|^2$.
9: **else** ▷ updating the LBG
10: Send actual gradient to the server: $\boldsymbol{\mu}_k^{(t)} \leftarrow \boldsymbol{g}_k^{(t)}$.
11: Update worker-copy of look-back gradient (LBG): $\boldsymbol{g}_k^{\ell} \leftarrow \boldsymbol{g}_k^{(t)}$.
12: **end if**

Global update at the aggregation server:

13: Initialize global parameter $\boldsymbol{\theta}^{(0)}$ and broadcast it across workers.
14: **for** $t = 0$ to $(T-1)$ **do**
15: Receive updates from workers $\{\boldsymbol{\mu}_k^{(t)}\}_{k=1}^{K}$.
16: Update global parameters: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \sum_{k=1}^{K} \omega_k \left[s_k \cdot \boldsymbol{\mu}_k^{(t)} \cdot \boldsymbol{g}_k^{\ell} + (1-s_k)\cdot \boldsymbol{\mu}_k^{(t)}\right]$,
where s_k is an indicator function given by, $s_k = \begin{cases} 1, & \text{if } \boldsymbol{\mu}_k^{(t)} \text{ is a scalar} \\ 0, & \text{otherwise, i.e., if } \boldsymbol{\mu}_k^{(t)} \text{ is a vector} \end{cases}$.
17: Update server-copy of look-back gradients (LBGs): $\boldsymbol{g}_k^{\ell} \leftarrow (1-s_k)\boldsymbol{\mu}_k^{(t)} + (s_k)\boldsymbol{g}_k^{\ell}, \ \forall k$.
18: **end for**

1. *Smoothness of Local Loss Functions:* Local loss function $F_k : \mathbb{R}^M \to \mathbb{R}, \forall k$, is β-smooth:
$$\left\|\nabla F_k(\boldsymbol{\theta}_x) - \nabla F_k(\boldsymbol{\theta}_y)\right\| \leq \beta \left\|\boldsymbol{\theta}_x - \boldsymbol{\theta}_y\right\|, \ \ \forall \boldsymbol{\theta}_x, \boldsymbol{\theta}_y \in \mathbb{R}^M. \tag{A1}$$

2. *SGD Characteristics:* Local gradients $\boldsymbol{g}_k(\boldsymbol{\theta}), \forall k$, estimated by SGD are unbiased estimators of the true gradients $\nabla F_k(\boldsymbol{\theta})$, and have a bounded variance $\sigma^2 \geq 0$; mathematically:
$$\mathbb{E}\left[\boldsymbol{g}_k(\boldsymbol{\theta})\right] = \nabla F_k(\boldsymbol{\theta}), \text{ and } \mathbb{E}\left[\|\boldsymbol{g}_k(\boldsymbol{\theta}) - \nabla F_k(\boldsymbol{\theta})\|^2\right] \leq \sigma^2, \ \ \forall \boldsymbol{\theta} \in \mathbb{R}^M. \tag{A2}$$

3. *Bounded Dissimilarity of Local Loss Functions:* For any realization of weights $\{\omega_k \geq 0\}_{k=1}^{K}$, where $\sum_{k=1}^{K}\omega_k = 1$, there exist non-negative constants $\Upsilon^2 \geq 1$ and $\Gamma^2 \geq 0$ such that
$$\sum_{k=1}^{K} \omega_k \|\nabla F_k(\boldsymbol{\theta})\|^2 \leq \Upsilon^2 \left\|\sum_{k=1}^{K}\omega_k \nabla F_k(\boldsymbol{\theta})\right\|^2 + \Gamma^2, \ \ \forall \boldsymbol{\theta} \in \mathbb{R}^M. \tag{A3}$$

Theorem 1. ***(General Convergence Characteristic of*** `LBGM`***)*** *Assume* **A1**, **A2**, **A3**, *and that* $\eta\beta \leq \min\left\{1/(2\tau), 1/\left(\tau\sqrt{2(1+4\Upsilon^2)}\right)\right\}$. *If the threshold value in step 7 of Algorithm 1 satisfies the*

condition $\delta_k^{\text{threshold}} \leq \Delta^2/\|\boldsymbol{d}_k^{(t)}\|^2$, $\forall k$ *(i.e., LBP error at step 6 satisfies* $\|\boldsymbol{d}_k^{(t)}\|^2 \sin^2(\alpha_k^{(t),\ell}) \leq \Delta^2$, $\forall k$*), where* $\Delta^2 \geq 0$ *is a constant, then after* T *rounds of global aggregations, the performance of* LBGM *is characterized by the following upper bound:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2\right] \leq \frac{8[F(\boldsymbol{\theta}^{(0)})-F^\star]}{\eta\tau T} + 16\Delta^2 + 8\eta\beta\sigma^2 + 5\eta^2\beta^2\sigma^2(\tau-1) + 20\eta^2\beta^2\Gamma^2\tau(\tau-1). \quad (3)$$

Proof. The proof is provided in Appendix A. ■

The condition on $\delta_k^{\text{threshold}}$ and LBP error $\sin^2(\alpha_k^{(t),\ell})$ in the above theorem implies that to have a fixed bound in (3), for a fixed Δ^2, a larger gradient norm $\|\boldsymbol{d}_k^{(t)}\|^2$ is associated with a tighter condition on LBP error $\sin^2(\alpha_k^{(t),\ell})$ and $\delta_k^{\text{threshold}}$. This is intuitive because a larger gradient norm corresponds to a larger estimation error when the gradient is recovered at the server for a given LBP (see Fig. 4). In practice, since the gradient norm $\|\boldsymbol{d}_k^{(t)}\|^2$ does not grow to infinity during model training, the condition on LBP error in Theorem 1, i.e., $\sin^2(\alpha_k^{(t),\ell}) \leq \Delta^2/\|\boldsymbol{d}_k^{(t)}\|^2$, can always be satisfied for any $\Delta^2 \geq 0$, since transmitting actual gradients of worker k makes $\alpha_k^{(t),\ell} = 0$. Given the general convergence behavior in Theorem 1, we next obtain specific choice of step size and an upper bound on Δ^2 for which LBGM approaches a stationary point of the global loss function (1).

Corollary 1. ***(Convergence of*** LBGM ***to a Stationary Point)*** *Assuming the conditions of Theorem 1, if* Δ^2 *introduced in Theorem 1 satisfies* $\Delta^2 \leq \eta$*, where* $\eta = 1/\sqrt{\tau T}$*, then* LBGM *converges to a stationary point of the global loss function, with the convergence bound characterized below:*

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2\right] \leq \mathcal{O}\left(\frac{1}{\sqrt{\tau T}}\right) + \mathcal{O}\left(\frac{\sigma^2}{\sqrt{\tau T}}\right) + \mathcal{O}\left(\frac{1}{\sqrt{\tau T}}\right) + \mathcal{O}\left(\frac{\sigma^2(\tau-1)}{\tau T}\right) + \mathcal{O}\left(\frac{(\tau-1)\Gamma^2}{T}\right). \quad (4)$$

Proof. The proof is provided in Appendix B. ■

Considering the definition of Δ^2 in Theorem 1 and the condition imposed on it in Corollary 1, the LBP error (i.e., $\sin^2(\alpha_k^{(t),\ell})$) should satisfy $\|\boldsymbol{d}_k^{(t)}\|^2 \sin^2(\alpha_k^{(t),\ell}) \leq \eta = 1/\sqrt{\tau T}$ for LBGM to reach a stationary point of the global loss. Since $\|\boldsymbol{d}_k^{(t)}\|^2$ is bounded during the model training, this condition on LBP error can always be satisfied by tuning the frequency of the full (actual) gradient transmission, i.e. updating the LBG as in lines 10&11 of Algorithm 1. Specifically, since the correlation across consecutive gradients is high and it drops as the gradients are sampled from distant epochs (see Fig. 3), a low value of LBP error can be obtained via more frequent LBG transmissions to the server, which makes the generated gradients similar to the stored LBG at the server.

Main Takeaways of Theorem 1, Corollary 1, and Algorithm 1:

1. *Recovering Vanilla-FL Bound:* In (3), if the LBGs are always propagated by all the devices, we have $\alpha_k^{(t),\ell} = 0$, $\forall k$, and thus $\Delta^2 = 0$ satisfies the condition on the LBP error. Then, (3) recovers the bound for vanilla FL (Wang et al., 2020a; Stich, 2019; Wang & Joshi, 2018).
2. *Recovering Centralized-SGD Bound:* In (3), if the LBGs are always propagated by all the devices, i.e., $\Delta^2 = 0$, the local dataset sizes are equal, i.e., $w_k = 1/K, \forall k$, and $\tau = 1$, then (3) recovers the bound for centralized-SGD, e.g., see Friedlander & Schmidt (2012).
3. *Unifying Algorithm 1 and Theorem 1:* The value of Δ^2 in (3) is determined by the value of the LBP error $\sin^2(\alpha_k^{(t),\ell})$, which is also reflected in step 7 of Algorithm 1. This suggests that the performance improves when the allowable threshold on $\sin^2(\alpha_k^{(t),\ell})$ is decreased (i.e., smaller Δ^2), which is the motivation behind introducing the tunable threshold $\delta_k^{\text{threshold}}$ in our algorithm.
4. *Effect of LPB Error on Convergence:* As the value of $\sin^2(\alpha_k^{(t),\ell})$ increases, it can be seen that the term in (3) containing Δ^2 will start diverging (it can become the same order as the gradient $\|\boldsymbol{d}_k^{(t)}\|^2$). In particular, the condition in Corollary 1 on Δ^2 avoids such scenario, under which the convergence to a stationary point is achieved.
5. *Performance vs. Communication Overhead Trade-off:* Considering step 7 of Algorithm 1, increasing the tolerable threshold on the LBP error increases the chance of transmitting a scalar (i.e., LBC) instead of the entire gradient to the server from each worker, leading to communication savings. However, as seen in Theorem 1 and the condition on Δ^2 in Corollary 1, the threshold on LBP error cannot be increased arbitrarily since the LBGM may show diverging behavior.

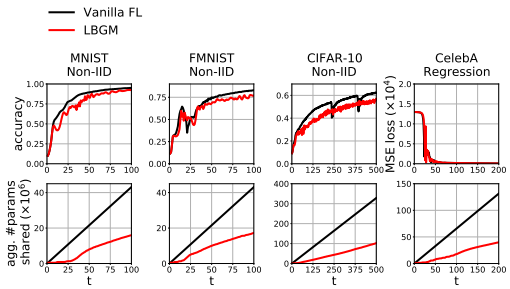

Figure 5: LBGM *as a Standalone Algorithm*. Irrespective of the dataset or data configuration across workers, LBGM consistently outperforms vanilla FL in terms of total number of parameters shared (bottom row) while achieving comparable accuracy (top row).

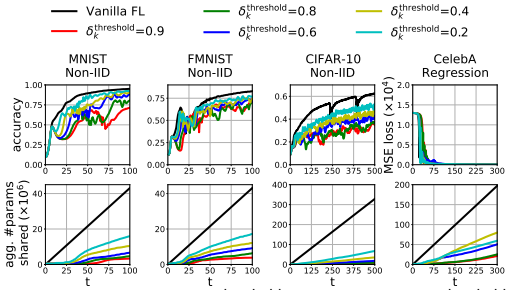

Figure 6: *Effect of* $\delta_k^{\text{threshold}}$ *on* LBGM. As $\delta_k^{\text{threshold}}$ decreases, the training may become unstable. For larger values of $\delta_k^{\text{threshold}}$, LBGM achieves communication benefits (bottom row) while maintaining a performance identical to vanilla FL (top row).

4 Experiments

We run experiments using several NN architectures and datasets. Specifically, we train: **S1:** CNN on FMNIST, MNIST, CelebA, and CIFAR-10, **S2:** FCN on FMNIST and MNIST, and **S3:** ResNet18 on CelebA, CIFAR-10 and CIFAR-100 for both iid and non-iid data distributions. We present results of **S1** (on non-iid data) in this section and defer the rest (including **S1** on iid data) to Appendix E.

Properties Studied. We specifically focus on four properties of LBGM: **P1:** benefits of gradient recycling by LBGM *as a standalone algorithm*, **P2:** confirming the insights from Theorem 1 about *effect of* $\delta_k^{\text{threshold}}$ on LBGM, **P3:** practical capabilities of LBGM *as a general plug-and-play algorithm* that can be stacked on top of other gradient compression techniques to further reduce communication overhead in FL training by exploiting the low-rank property of the gradient subspace, and finally **P4:** generalizability of LBGM *to other distributed learning* frameworks, e.g., multi-processor ML system over multiple GPUs within a single machine, or even multiple machines linked together.

Baselines. For **P1**, we compare standalone LBGM with vanilla FL. For **P3**, we stack LBGM on top of existing gradient compression techniques: top-K and ATOMO (Wang et al., 2018) – among the state-of-the-art techniques in the domains of sparsification and low-rank approximation based gradient compression, respectively. For **P4**, we stack LBGM on top of SignSGD (Bernstein et al., 2018), which is one of the state-of-the-art solutions in gradient compression for distributed learning.

Implementation Details. We consider a FL system consisting of 100 workers. We consider both the independently and identically distributed (iid) and non-iid data distributions among the workers. Under the iid setting, each worker has training data with all the labels, while under the non-iid setting the workers have training data only from a subset of all labels (e.g., each worker has training data from only 3 of the 10 classes in MNIST/FMNIST). The workers train identical networks with a mini-batch size ranging from 128 to 512 based on the choice of dataset. We implement LBGM with uniform $\delta_k^{\text{threshold}}$ across workers. We also use error feedback (Karimireddy et al., 2019) as standard only if top-K sparsification is used in the training. The FL system is simulated using PyTorch (Paszke et al., 2019) and PySyft (Ryffel et al., 2018) and trained on a 48GB Tesla-P100 GPU with 128GB RAM. All of our code, and hyperparameters are available at https://github.com/PairML/LBGM.

Complexity. Compared to other gradient compression techniques, the processing overhead introduced by LBGM is negligible. Considering Algorithm 1, the calculation of LBCs/LBP errors involves inner products and division of scalars, while reconstruction of LBG-based gradient approximations at the server is no more expensive than the global aggregation step: since the global aggregation step requires averaging of local model parameters, it can be combined with gradient reconstruction. This also holds for LBGM as a plug-and-play algorithm, as top-K and ATOMO (Wang et al., 2018) introduce considerable computation overhead. In particular, LBGM has $\mathcal{O}(M)$ complexity, where M is the dimension of the NN parameter, which is inexpensive to plug on top of top-K ($\mathcal{O}(M \log M)$), ATOMO (Wang et al., 2018) ($\mathcal{O}(M^2)$), and SignSGD (Bernstein et al., 2018) ($\mathcal{O}(M)$) methods.

LBGM **as a Standalone Algorithm.** We evaluate the effect of gradient recycling by LBGM in FL. We choose $\delta_k^{\text{threshold}} = 0.2, \ \forall k$, and observe that LBGM is able to save communication overhead in the order of 10^7 floating point parameters per worker. Communication benefits are depicted in Fig. 5. Refer to Appendix E.1 for similar results on other datasets and NN architectures.

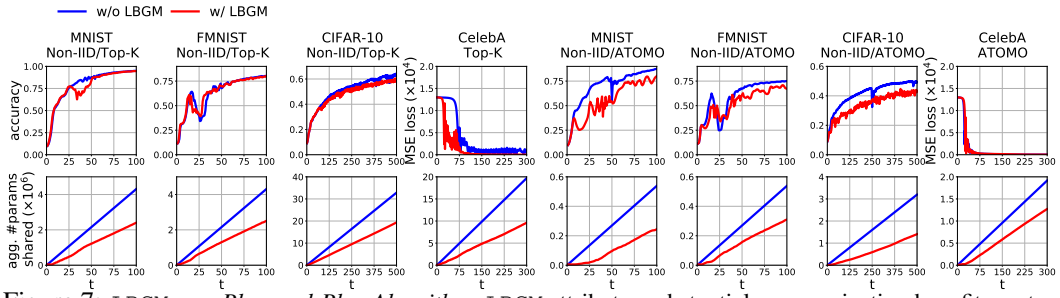

Figure 7: LBGM *as a Plug-and-Play Algorithm.* LBGM attributes substantial communication benefits on top of existing gradient compression techniques by exploiting the rank-characteristics of gradient-space. While both Top-K and ATOMO are known to achieve state-of-the-art performance of their respective domains of sparsification and low-rank approximation, plugging LBGM adds notable improvements to these techniques.

Effect of $\delta_k^{\text{threshold}}$. We analyze the behavior of LBGM under different $\delta_k^{\text{threshold}}$ values. As pointed out in takeaway 3 in Sec. 3, decreasing the allowable threshold on LBP error, $\sin^2(\alpha_k^{(t),\ell})$ improves the model performance. The effect of changing the threshold is controlled by changing $\delta_k^{\text{threshold}}$ values in Algorithm 1. A drop in model performance can be observed as we increase $\delta_k^{\text{threshold}}$ in Fig. 6. This performance drop is accompanied by an increase in communication savings as discussed in takeaway 5 in Sec. 3, i.e., while a higher threshold value requires less frequent updates of LBGs, it converges slower or to a suboptimal solution. Refer to Appendix E.2 for additional results.

LBGM **as a Plug-and-Play Algorithm.** Next we show that state-of-the-art results in gradient compression can benefit from using LBGM. For plug-and-play setup, the LBGM follows the same steps as in Algorithm 1, with the slight modification that the output of gradient compression techniques, e.g., top-K, ATOMO, are used in place of accumulated gradient $\boldsymbol{g}_k^{(t)}$ and LBGs $\boldsymbol{g}_k^{\ell}$, $\forall k$. We consider top-K sparsification, and ATOMO approximation as the base gradient compression techniques and plug LBGM to operate on their outputs. Interestingly, LBGM adds on top of existing communication benefits of both top-K and ATOMO in the order of 10^6 and 10^5 floating point model parameters per worker, respectively, as shown in Fig. 7: while top-K and ATOMO compress gradients through approximation, they do not alter the underlying low-rank characteristics of the gradient-space. Our algorithm LBGM exploits this property to obtain substantial communication savings on top of that of these algorithms. Refer to Appendix E.3 for additional experiments.

Generalizability of LBGM to Distributed Training. Although throughout the paper, we focus on FL, LBGM can be applied to more general distributed gradient computation settings, e.g., multi-core, multi-GPU ML systems. Thus, most of our experiments above can be easily extended to more general distributed settings. While non-iid distribution is not a consideration in multi-core distributed systems because of possibility of data transfer, there is a research interest to minimize parameter exchange among systems to reduce communication latency. In the domain of distributed training, SignSGD (Bernstein et al., 2018) is one of the state-of-the-art solution that reduces the communication requirements by several order of magnitude by converting floating-point parameter communication to binary bit communication. We apply LBGM as a plug-and-play addition on top of SignSGD and find that LBGM can further reduce the overall bits transferred by SignSGD, by an order of 10^7 bits in Fig. 8. Refer to Appendix E.4 for additional experiments.

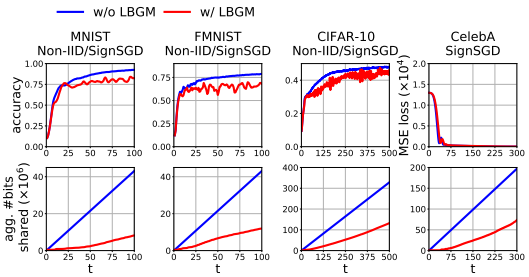

Figure 8: Application of LBGM as a plug-and-play algorithm with SignSGD in distributed training.

5 Related Work

Eigenvalues of Hessian. While to the best of our knowledge there are no prior works directly studying the gradient subspace, there exist several works (Sagun et al., 2016; 2017; Gur-Ari et al., 2018) that analyze the eigenvalues of the Hessian to quantify the overparameterization of NNs. (Sagun et al., 2016; 2017) divide the eigenspace of Hessian into two parts: bulk (with eigenvalues

concentrated around zero) and edges (with eigenvalues scattered away from zero), and show that increasing the complexity of network only affects the bulk component. (Gur-Ari et al., 2018) aims to show that the cardinality of the edge space is equal to the number of classes in the classification task. In parallel, (Ghorbani et al., 2019) studies the Hessian of larger networks and argues that the existence of large isolated eigenvalues in eigenspace of the Hessian is correlated with the slow convergence.

Gradient Compression. Gradient compression techniques proposed in literature can be broadly categorized into (i) sparsification, (ii) quantization, and (iii) low-rank approximation. Sparsification compresses gradients by promoting sparsity in transmitted gradients. Specifically, (Wangni et al., 2018) proposes probabilistic sparsification by optimizing the trade-off between sparsity and gradient variance. (Sattler et al., 2019) extends top-K sparsification using ternarization for downstream communication. Quantization methods on the other hand aim to find approximate representations of gradients that promote tensor compression. (Seide et al., 2014; Alistarh et al., 2017) propose approximation using quantization with error feedback to substantially boost communication compression. These works were originally proposed for inter-GPU communication speedups and not FL. Finally, low-rank approximation techniques (Wang et al., 2018; Vogels et al., 2019) decompose large gradients matrices as an outer product of smaller matrices to reduce communication cost.

Communication Delay. Another line of work aiming to reduce communication cost proposes methods that decrease the gradient communication frequency. (Hosseinalipour et al., 2020; Lin et al., 2021) propose device-to-device local network communication, while SloMo (Wang et al., 2020b) proposes momentum-based updates after delayed aggregation to reduce overall communications.

Model Compression. Compressing model would naturally reduce communication overhead. (Li et al., 2020) extends the lottery ticket hypothesis to federated learning in the interest of communication saving. There are several other model compression works among centralized training. Both model distillation (Ba & Caruana, 2014; Hinton et al., 2015) that train smaller networks to mimic a large network, and model pruning (LeCun et al., 1990; Hinton et al., 2015) that systematically removes model parameters exploit the fact that NNs tend to be dramatically overparameterized in practice and neurons can be removed without affecting the accuracy. Squeezenets (Iandola et al., 2016) and Mobilenets (Howard et al., 2017) are designed with orders of magnitude smaller size than conventional architectures. (Denil et al., 2013) represents model parameters with low rank matrices.

Unlike prior works that study the low-rank property in the context of gradient compression during a single gradient transfer step (via exploiting low-rank approximation of matrices), our work explores the low rank property by studying the relationship among gradients generated at successive gradient transfer steps during the distributed training process (i.e., low-rank property of subspace spanned by these gradients). To the best our knowledge, we are the first to study this effect of "overparameterization" on optimization of NN via the rank-characteristics of the first order optimization (gradient) landscape. We propose the `LBGM` algorithm based on the low-rank property of gradient-space, which is itself orthogonal to other related works in the field of gradient compression, as none of these techniques exploit the rank-property of the gradient space. Thus, most of the prior techniques would further benefit by using `LBGM` algorithm during FL training as we show in our experiments for top-K, ATOMO, and SignSGD. Our work gives a novel insight into designing a class of techniques based on "Look-back Gradients" that can be used in general distributed machine learning systems.

6 Discussion & Conclusions

In this paper, we gave a fresh perspective into the effect "overparamterization" on the optimization of NNs by analyzing the gradient subspace. We drew insights from our preliminary experiments to support our two hypotheses: *"the subspace spanned by gradients generated during SGD is low-rank"* and *"principal gradients can be approximated using a subset of actual gradients"*. We then proposed `LBGM` algorithm that uses the low-rank property of gradient-space to recycle the previously generated gradients at the devices to represent the newly generated gradients. `LBGM` reduces communication overhead in FL by several orders of magnitude by replacing the transmission of vectors of million/billion parameters with a single scalar at each device. We theoretically characterized the convergence behavior of `LBGM` algorithm and experimentally substantiate our claims. Furthermore, we showed that our algorithm `LBGM` can be extended to further reduce latency of communication in large distributed training systems by plugging `LBGM` on top of other gradient compression techniques.

REFERENCES

Turing-NLG: A 17-billion-parameter language model by Microsoft. https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/, 2020.

Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Sheikh Shams Azam, Taejin Kim, Seyyedali Hosseinalipour, Christopher Brinton, Carlee Joe-Wong, and Saurabh Bagchi. Towards Generalized and Distributed Privacy-preserving Representation Learning. *arXiv preprint arXiv:2010.01792*, 2021.

Lei Jimmy Ba and Rich Caruana. Do Deep Nets really need to be Deep? In *Neural Information Processing Systems (NeurIPS)*, 2014.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. SignSGD: Compressed Optimisation for Non-convex Problems. In *International Conference on Machine Learning (ICML)*, 2018.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Corinna Cortes and Vladimir Vapnik. Support-vector Networks. *Machine Learning*, 20(3):273–297, 1995.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting Parameters in Deep Learning. In *Neural Information Processing Systems (NeurIPS)*, 2013.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision (IJCV)*, 88(2):303–338, 2010.

Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations (ICLR)*, 2019.

Michael P Friedlander and Mark Schmidt. Hybrid Deterministic-stochastic Methods for Data Fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An Investigation into Neural Net Optimization via Hessian Eigenvalue Density. In *International Conference on Machine Learning (ICML)*, pp. 2232–2241. PMLR, 2019.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient Descent Happens in a Tiny Subspace. *arXiv preprint arXiv:1812.04754*, 2018.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both Weights and Connections for Efficient Neural Networks. In *Neural Information Processing Systems (NeurIPS)*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2961–2969, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

Seyyedali Hosseinalipour, Sheikh Shams Azam, Christopher G Brinton, Nicolo Michelusi, Vaneet Aggarwal, David J Love, and Huaiyu Dai. Multi-stage Hybrid Federated Learning over Large-scale Wireless Fog Networks. *arXiv preprint arXiv:2007.09511*, 2020.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4700–4708, 2017.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and< 0.5 MB Model Size. *arXiv preprint arXiv:1602.07360*, 2016.

Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error Feedback fixes SignSGD and other Gradient Compression Schemes. In *International Conference on Machine Learning (ICML)*, 2019.

Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated Learning: Strategies for Improving Communication Efficiency. In *Neural Information Processing Systems (NeurIPS)*, 2016.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Neural Information Processing Systems (NeurIPS)*, volume 25, pp. 1097–1105, 2012.

Yann LeCun and Corinna Cortes. MNIST Handwritten Digit Database, 2010. URL http://yann.lecun.com/exdb/mnist/.

Yann LeCun, John S Denker, and Sara A Solla. Optimal Brain Damage. In *Neural Information Processing Systems (NeurIPS)*, pp. 598–605, 1990.

Ang Li, Jingwei Sun, Binghui Wang, Lin Duan, Sicheng Li, Yiran Chen, and Hai Li. LotteryFL: Personalized and Communication-efficient Federated Learning with Lottery Ticket Hypothesis on Non-iid Datasets. *arXiv preprint arXiv:2008.03371*, 2020.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient Convnets. In *International Conference on Learning Representations (ICLR)*, 2017.

Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the Convergence of FedAvg on Non-IID Data. In *International Conference on Learning Representations (ICLR)*.

Frank Po-Chen Lin, Seyyedali Hosseinalipour, Sheikh Shams Azam, Christopher G Brinton, and Nicolo Michelusi. Two Timescale Hybrid Federated Learning with Cooperative D2D Local Model Aggregations. *arXiv preprint arXiv:2103.10481*, 2021.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV)*, pp. 740–755, 2014.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot Multibox Detector. In *European Conference on Computer Vision (ECCV)*, pp. 21–37. Springer, 2016.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692*, 2019a.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the Value of Network Pruning. In *International Conference on Learning Representations (ICLR)*, 2019b.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep Learning Face Attributes in the Wild. In *IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1273–1282. PMLR, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Neural Information Processing Systems (NeurIPS)*. 2019.

Karl Pearson. LIII. On Lines and Planes of Closest Fit to Systems of Points in Space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 1(8):9, 2019.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *Neural Information Processing Systems (NeurIPS)*, 2015.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pp. 234–241, 2015.

Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A Generic Framework for Privacy Preserving Deep Learning. *arXiv preprint arXiv:1811.04017*, 2018.

Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the Hessian in Deep Learning: Singularity and Beyond. *arXiv preprint arXiv:1611.07476*, 2016.

Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical Analysis of the Hessian of Over-parametrized Neural Networks. *arXiv preprint arXiv:1706.04454*, 2017.

Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and Communication-efficient Federated Learning from Non-iid Data. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3400–3413, 2019.

Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit Stochastic Gradient Descent and its Application to Data-parallel Distributed Training of Speech DNNs. In *Conference of the International Speech Communication Association (INTERSPEECH)*, 2014.

Reza Shokri and Vitaly Shmatikov. Privacy-preserving Deep Learning. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1310–1321, 2015.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Sebastian U Stich. Local SGD Converges Fast and Communicates Little. In *International Conference on Learning Representations (ICLR)*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. PowerSGD: Practical Low-rank Gradient Compression for Distributed Optimization. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Hongyi Wang, Scott Sievert, Zachary Charles, Shengchao Liu, Stephen Wright, and Dimitris Papailiopoulos. ATOMO: Communication-efficient Learning via Atomic Sparsification. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Jianyu Wang and Gauri Joshi. Cooperative SGD: A Unified Framework for the Design and Analysis of Communication-efficient SGD Algorithms. *arXiv preprint arXiv:1808.07576*, 2018.

Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. In *Neural Information Processing Systems (NeurIPS)*, 2020a.

Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael Rabbat. SlowMo: Improving Communication-efficient Distributed SGD with Slow Momentum. In *International Conference on Learning Representations (ICLR)*, 2020b.

Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient Sparsification for Communication-efficient Distributed Optimization. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Jingfeng Wu, Wenqing Hu, Haoyi Xiong, Jun Huan, Vladimir Braverman, and Zhanxing Zhu. On the Noisy Gradient Descent that Generalizes as SGD. In *International Conference on Machine Learning (ICML)*, pp. 10367–10376. PMLR, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, 2017.

Appendix

Table of Contents

A Proof of Theorem 1 **14**
A.1 Condition on Learning Rate 20

B Proof of Corollary 1 **21**

C Additional Pseudocodes **22**

D Additional Preliminary Experiments **23**
D.1 PCA Component Progression 23
D.2 Overlap of Actual and Principal Gradient 23
D.3 Similarity among Consecutive Generated Gradients 24

E Additional LBGM Experiments **24**
E.1 LBGM as a Standalone Algorithm. 25
E.2 Effect of $\delta_k^{\text{threshold}}$ on LBGM. 25
E.3 LBGM as a Plug-and-Play Algorithm. 25
E.4 Generalizability of LBGM to Distributed Training. 25

A PROOF OF THEOREM 1

We start by introducing a lemma, that is used prominently throughout the analysis that follows.

Lemma 1. *For a sequence of vectors* $\{\boldsymbol{a}_i\}_{i=1}^N$*, such that* $\mathbb{E}\left[\boldsymbol{a}_i|\boldsymbol{a}_{i-1}, \boldsymbol{a}_{i-2}, \cdots, \boldsymbol{a}_1\right] = \mathbf{0}, \forall i,$

$$\mathbb{E}\left[\left\|\sum_{i=1}^N \boldsymbol{a}_i\right\|^2\right] = \sum_{i=1}^N \mathbb{E}\left[\|\boldsymbol{a}_i\|^2\right]. \tag{L2}$$

Proof.

$$\mathbb{E}\left[\left\|\sum_{i=1}^N \boldsymbol{a}_i\right\|^2\right] = \sum_{i=1}^N \mathbb{E}\left[\|\boldsymbol{a}_i\|^2\right] + \sum_{i=1}^N \sum_{\substack{j=1 \\ j\neq i}}^N \mathbb{E}\left[\boldsymbol{a}_i^\top \boldsymbol{a}_j\right]. \tag{5}$$

Using the law of total expectation, assuming $i < j$, we get

$$\mathbb{E}\left[\boldsymbol{a}_i^\top \boldsymbol{a}_j\right] = \mathbb{E}\left[\boldsymbol{a}_i^\top \mathbb{E}\left[\boldsymbol{a}_j|\boldsymbol{a}_i, \cdots, \boldsymbol{a}_1\right]\right] = \mathbf{0}, \tag{6}$$

which completes the proof. ■

Next, we define a few auxiliary variables that would be referenced in the proof later: as defined in the main text, $\boldsymbol{g}_k^{(t)} = \sum_{b=0}^{\tau-1} g_k(\boldsymbol{\theta}_k^{(t,b)})$ is the accumulated stochastic gradient at worker k, where b ranging from 0 to $\tau - 1$ denotes the rounds of local updates. Using Assumption (**A2**), error in stochastic gradient approximation $g_k(\boldsymbol{\theta}_k^{(t,b)})$ can be defined as $\boldsymbol{\epsilon}_k^{(t,b)} = g_k(\boldsymbol{\theta}_k^{(t,b)}) - \nabla F_k(\boldsymbol{\theta}_k^{(t,b)})$. Consequently, we can write the stochastic gradient as $g_k(\boldsymbol{\theta}_k^{(t,b)}) = \nabla F_k(\boldsymbol{\theta}_k^{(t,b)}) + \boldsymbol{\epsilon}_k^{(t,b)}$ where $\nabla F_k(\boldsymbol{\theta}_k^{(t,b)})$ is the true gradient. From Assumption (**A2**) it follows that

$$\mathbb{E}\left[\boldsymbol{\epsilon}_k^{(t,b)}\right] = \mathbf{0} \text{ and } \mathbb{E}\left[\left\|\boldsymbol{\epsilon}_k^{(t,b)}\right\|^2\right] \leq \sigma^2. \tag{7}$$

We also introduce the normalized stochastic gradient $\boldsymbol{d}_k^{(t)}$ given by

$$\boldsymbol{d}_k^{(t)} = \frac{\boldsymbol{g}_k^{(t)}}{\tau} = \frac{1}{\tau}\sum_{b=0}^{\tau-1} \mathbf{g}_k(\boldsymbol{\theta}_k^{(t,b)}) = \frac{1}{\tau}\sum_{b=0}^{\tau-1} \nabla F_k(\boldsymbol{\theta}_k^{(t,b)}) + \frac{1}{\tau}\sum_{b=0}^{\tau-1} \boldsymbol{\epsilon}_k^{(t,b)} = \boldsymbol{h}_k^{(t)} + \boldsymbol{\epsilon}_k^{(t)}, \tag{8}$$

where we define

$$\text{Cumulative Average of the true gradient: } \boldsymbol{h}_k^{(t)} = \frac{1}{\tau}\sum_{b=0}^{\tau-1} \nabla F_k(\boldsymbol{\theta}_k^{(t,b)}), \text{ and} \tag{9}$$

$$\text{Cumulative Average of the SGD error: } \boldsymbol{\epsilon}_k^{(t)} = \frac{1}{\tau}\sum_{b=0}^{\tau-1} \boldsymbol{\epsilon}_k^{(t,b)}. \tag{10}$$

Next, we evaluate the first and second moment of normalized SGD error, $\boldsymbol{\epsilon}_k^{(t)}$ as follows

$$\mathbb{E}\left[\boldsymbol{\epsilon}_k^{(t)}\right] = \mathbb{E}\left[\frac{1}{\tau}\sum_{b=0}^{\tau-1} \boldsymbol{\epsilon}_k^{(t,b)}\right] = \frac{1}{\tau}\sum_{b=0}^{\tau-1} \mathbb{E}\left[\boldsymbol{\epsilon}_k^{(t,b)}\right] = \mathbf{0}, \tag{11}$$

$$\mathbb{E}\left[\left\|\boldsymbol{\epsilon}_k^{(t)}\right\|^2\right] = \mathbb{E}\left[\left\|\frac{1}{\tau}\sum_{b=0}^{\tau-1} \boldsymbol{\epsilon}_k^{(t,b)}\right\|^2\right] = \frac{1}{\tau^2}\sum_{b=0}^{\tau-1} \mathbb{E}\left[\left\|\boldsymbol{\epsilon}_k^{(t,b)}\right\|^2\right] \leq \frac{\sigma^2}{\tau}, \tag{12}$$

where (11) uses Assumption (**A2**) and (12) uses Lemma 1.

Now we can proceed to the proof of Theorem 1. We start with the update rule of `LBGM`, where a round of global update is given by

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \sum_{k=1}^K \omega_k \widetilde{\boldsymbol{g}}_k^{(t)} = \boldsymbol{\theta}^{(t)} - \tau\eta \sum_{k=1}^K \omega_k \widetilde{\boldsymbol{d}}_k^{(t)}, \tag{13}$$

where $\widetilde{\boldsymbol{g}}_k^{(t)}$ is the approximate gradient shared by worker k with the server and $\widetilde{\boldsymbol{d}}_k^{(t)}$ is given by

$$\widetilde{\boldsymbol{d}}_k^{(t)} = \widetilde{\boldsymbol{g}}_k^{(t)}/\tau = \rho_k^{(t),\ell}\boldsymbol{g}_k^{\ell}/\tau = \rho_k^{(t),\ell}\boldsymbol{d}_k^{\ell} \tag{14}$$

where $\boldsymbol{d}_k^{\ell} = \boldsymbol{g}_k^{\ell}/\tau$ is the normalized stochastic gradient w.r.t. the last LBG shared by worker k. Also, from the trigonometric relationship shown in Fig. 4, we have,

$$\left\|\rho_k^{(t),\ell}\boldsymbol{d}_k^{\ell}\right\| = \left\|\boldsymbol{d}_k^{(t)}\cos(\alpha_k^{(t),\ell})\right\| \tag{15}$$

Similar to $\boldsymbol{d}_k^{(t)}$ from (8), $\boldsymbol{d}_k^{\ell}$ can be split into the cumulative average of true look-back gradient $\boldsymbol{h}_k^{\ell}$ and the corresponding cumulative average of SGD error $\boldsymbol{\epsilon}_k^{\ell}$, given by

$$\boldsymbol{d}_k^{\ell} = \boldsymbol{h}_k^{\ell} + \boldsymbol{\epsilon}_k^{\ell}. \tag{16}$$

From Assumption (**A1**), the global loss function is β smooth, which implies:

$$F(\boldsymbol{\theta}^{(t+1)}) - F(\boldsymbol{\theta}^{(t)}) \leq \left\langle \nabla F(\boldsymbol{\theta}^{(t)}), \boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)} \right\rangle + \frac{\beta}{2}\left\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\right\|^2. \tag{17}$$

Taking expectation over $\boldsymbol{\epsilon}_k^{(t,b)}, k \in \{1, 2, \cdots, K\}, b \in \{0, 1, \cdots, \tau - 1\}$,

$$\begin{aligned}
&\mathbb{E}\left[F(\boldsymbol{\theta}^{(t+1)})\right] - F(\boldsymbol{\theta}^{(t)}) \\
&\qquad \leq -\tau\eta\, \underbrace{\mathbb{E}\left[\left\langle \nabla F(\boldsymbol{\theta}^{(t)}), \sum_{k=1}^{K}\omega_k \cdot \widetilde{\boldsymbol{d}}_k^{(t)} \right\rangle\right]}_{\mathbf{Z_1}} + \frac{\tau^2\eta^2\beta}{2}\, \underbrace{\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\widetilde{\boldsymbol{d}}_k^{(t)}\right\|^2\right]}_{\mathbf{Z_2}}.
\end{aligned} \tag{18}$$

We evaluate $\mathbf{Z_1}$ as follows

$$\begin{aligned}
\mathbf{Z_1} &= \mathbb{E}\left[\left\langle \nabla F(\boldsymbol{\theta}^{(t)}), \sum_{k=1}^{K}\omega_k\widetilde{\boldsymbol{d}}_k^{(t)} \right\rangle\right] = \mathbb{E}\left[\left\langle \nabla F(\boldsymbol{\theta}^{(t)}), \sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \boldsymbol{d}_k^{(t)} + \widetilde{\boldsymbol{d}}_k^{(t)}\right) \right\rangle\right] \\
&= \underbrace{\mathbb{E}\left[\left\langle \nabla F(\boldsymbol{\theta}^{(t)}), \sum_{k=1}^{K}\omega_k\boldsymbol{d}_k^{(t)} \right\rangle\right]}_{\mathbf{Z_{1,1}}} - \underbrace{\mathbb{E}\left[\left\langle \nabla F(\boldsymbol{\theta}^{(t)}), \sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right) \right\rangle\right]}_{\mathbf{Z_{1,2}}},
\end{aligned} \tag{19}$$

where $\mathbf{Z_{1,1}}$ is given by,

$$\begin{aligned}
\mathbf{Z_{1,1}} &\overset{(i)}{=} \mathbb{E}\left[\left\langle \nabla F(\boldsymbol{\theta}^{(t)}), \sum_{k=1}^{K}\omega_k\boldsymbol{h}_k^{(t)} \right\rangle\right] + \mathbb{E}\left[\left\langle \nabla F(\boldsymbol{\theta}^{(t)}), \sum_{k=1}^{K}\omega_k\boldsymbol{\epsilon}_k^{(t)} \right\rangle\right] \\
&\overset{(ii)}{=} \frac{1}{2}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + \frac{1}{2}\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\boldsymbol{h}_k^{(t)}\right\|^2\right] - \frac{1}{2}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t)}) - \sum_{k=1}^{K}\omega_k\boldsymbol{h}_k^{(t)}\right\|^2\right],
\end{aligned} \tag{20}$$

where (i) follows from (16) and (ii) uses $2\langle \mathbf{a}, \mathbf{b}\rangle = \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2 - \|\mathbf{a} - \mathbf{b}\|^2$ for any two real vectors $\mathbf{a}$ and $\mathbf{b}$. We next upper bound the term $\mathbf{Z_{1,2}}$ (although $Z_{1,2}$ has a negative sign in (19), Z_1 also appears with a negative sign in (18) which allows us to do the upper bound) as follows

$$\begin{aligned}
\mathbf{Z_{1,2}} &= \mathbb{E}\left[\left\langle \nabla F(\boldsymbol{\theta}^{(t)}), \sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right) \right\rangle\right] \\
&\overset{(i)}{\leq} \frac{1}{4}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + \mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right)\right\|^2\right],
\end{aligned} \tag{21}$$

where (i) follows from $\langle a, b\rangle \leq (1/4)\|a\|^2 + \|b\|^2$ (result of Cauchy-Schwartz and Young's inequalities). Substituting (20) and (21) back in (19), we get

$$-\mathbf{Z_1} \leq -\frac{1}{4}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 - \frac{1}{2}\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right] + \frac{1}{2}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t)}) - \sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right]$$
$$+ \mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right)\right\|^2\right]. \tag{22}$$

We next bound the term $\mathbf{Z_2}$ in (18) as follows

$$\mathbf{Z_2} = \mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \widetilde{\boldsymbol{d}}_k^{(t)}\right\|^2\right] = \mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \boldsymbol{d}_k^{(t)} + \widetilde{\boldsymbol{d}}_k^{(t)}\right)\right\|^2\right]$$
$$\overset{(i)}{\leq} \underbrace{2\,\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{d}_k^{(t)}\right\|^2\right]}_{\mathbf{Z_{2,1}}} + 2\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right)\right\|^2\right], \tag{23}$$

where $\mathbf{Z_{2,1}}$ is given by,

$$\mathbf{Z_{2,1}} = \mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{d}_k^{(t)}\right\|^2\right] \overset{(i)}{=} \mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right] + \mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{\epsilon}_k^{(t)}\right\|^2\right]$$
$$\overset{(ii)}{\leq} \mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right] + \sum_{k=1}^{K}\omega_k \mathbb{E}\left[\left\|\boldsymbol{\epsilon}_k^{(t)}\right\|^2\right] \overset{(iii)}{=} \mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right] + \frac{\sigma^2}{\tau}, \tag{24}$$

where (i) follows from (8), (ii) uses Jensen's inequality: $\|\sum_{k=1}^{K}\omega_k \boldsymbol{a}_k\|^2 \leq \sum_{k=1}^{K}\omega_k \|\mathbf{a}_k\|^2$, s.t. $\sum_{k=1}^{K}\omega_k = 1$, and (iii) uses (12). Plugging (24) back into (23), we get

$$\mathbf{Z_2} \leq 2\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right] + \frac{2\sigma^2}{\tau} + 2\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right)\right\|^2\right]. \tag{25}$$

Substituting (22) and (25) back in (18), we get

$$\mathbb{E}\left[F(\boldsymbol{\theta}^{(t+1)})\right] - F(\boldsymbol{\theta}^{(t)}) \leq -\frac{\tau\eta}{4}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 - \frac{\tau\eta}{2}\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right]$$
$$+ \frac{\tau\eta}{2}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t)}) - \sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right] + \tau\eta\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right)\right\|^2\right]$$
$$+ \tau^2\eta^2\beta\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right] + \tau\eta^2\beta\sigma^2 + \tau^2\eta^2\beta\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right)\right\|^2\right]$$
$$= -\frac{\tau\eta}{4}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 - \frac{\tau\eta}{2}(1 - 2\tau\eta\beta)\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right] + \frac{\tau\eta}{2}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t)}) - \sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right]$$
$$+ \tau\eta(1 + \tau\eta\beta)\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right)\right\|^2\right] + \tau\eta^2\beta\sigma^2. \tag{26}$$

Choosing $\tau\eta\beta \leq 1/2$, implies that $-\frac{\tau\eta}{2}(1-2\tau\eta\beta) \leq 0$ and $1+\tau\eta\beta \leq 3/2 < 2$, which results in simplification of (26) as:

$$\begin{aligned}
\frac{\mathbb{E}\left[F(\boldsymbol{\theta}^{(t+1)})\right] - F(\boldsymbol{\theta}^{(t)})}{\eta\tau} &\leq -\frac{1}{4}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + \frac{1}{2}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t)}) - \sum_{k=1}^{K}\omega_k \boldsymbol{h}_k^{(t)}\right\|^2\right] \\
&\quad + 2\mathbb{E}\left[\left\|\sum_{k=1}^{K}\omega_k\left(\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right)\right\|^2\right] + \eta\beta\sigma^2 \\
&\overset{(i)}{\leq} -\frac{1}{4}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + \frac{1}{2}\sum_{k=1}^{K}\omega_k\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t)}) - \boldsymbol{h}_k^{(t)}\right\|^2\right] \\
&\quad + 2\sum_{k=1}^{K}\omega_k \underbrace{\mathbb{E}\left[\left\|\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right\|^2\right]}_{\mathbf{Z_3}} + \eta\beta\sigma^2,
\end{aligned} \tag{27}$$

where (i) follows from Jensen's inequality $\|\sum_{k=1}^{K}\omega_k \boldsymbol{a}_k\|^2 \leq \sum_{k=1}^{K}\omega_k\|\mathbf{a}_k\|^2$, s.t. $\sum_{k=1}^{K}\omega_k = 1$ and $\boldsymbol{\theta}_k^{(t)} = \boldsymbol{\theta}^{(t)}, \forall k$ due to local synchronization. Now $\mathbf{Z_3}$ can be bounded as follows:

$$\begin{aligned}
\mathbf{Z_3} &= \mathbb{E}\left[\left\|\boldsymbol{d}_k^{(t)} - \widetilde{\boldsymbol{d}}_k^{(t)}\right\|^2\right] \overset{(i)}{=} \mathbb{E}\left[\left\|\boldsymbol{d}_k^{(t)} - \rho_k^{(t),\ell}\boldsymbol{d}_k^{\ell}\right\|^2\right] \overset{(ii)}{=} \mathbb{E}\left[\left\|\boldsymbol{d}_k^{(t)} - \frac{\frac{1}{\tau^2}\left\langle \boldsymbol{g}_k^{(t)}, \boldsymbol{g}_k^{\ell}\right\rangle}{\frac{1}{\tau^2}\left\|\boldsymbol{g}_k^{\ell}\right\|^2}\boldsymbol{d}_k^{\ell}\right\|^2\right] \\
&\overset{(iii)}{=} \mathbb{E}\left[\left\|\boldsymbol{d}_k^{(t)} - \frac{\left\langle \boldsymbol{d}_k^{(t)}, \boldsymbol{d}_k^{\ell}\right\rangle}{\left\|\boldsymbol{d}_k^{\ell}\right\|^2}\boldsymbol{d}_k^{\ell}\right\|^2\right] = \mathbb{E}\left[\left\|\boldsymbol{d}_k^{(t)}\right\|^2 - \frac{\left\langle \boldsymbol{d}_k^{(t)}, \boldsymbol{d}_k^{\ell}\right\rangle^2}{\left\|\boldsymbol{d}_k^{\ell}\right\|^2}\right] \\
&\overset{(iv)}{=} \mathbb{E}\left[\left\|\boldsymbol{d}_k^{(t)}\right\|^2 - \left\|\boldsymbol{d}_k^{(t)}\right\|^2\cos^2(\boldsymbol{\alpha}_k^{(t),\ell})\right] = \mathbb{E}\left[\left\|\boldsymbol{d}_k^{(t)}\right\|^2\left(1 - \cos^2(\boldsymbol{\alpha}_k^{(t),\ell})\right)\right] \\
&= 2\mathbb{E}\left[\left\|\boldsymbol{d}_k^{(t)}\right\|^2\sin^2(\boldsymbol{\alpha}_k^{(t),\ell})\right] \overset{(v)}{\leq} \mathbb{E}\left[\Delta^2\right] = \Delta^2,
\end{aligned} \tag{28}$$

where (i) uses (14), (ii) uses LBGM definition from (**D1**), (iii) uses the fact that $\boldsymbol{d}_k^{(t)} = \boldsymbol{g}_k^{\ell}/\tau$, (iv) uses $\langle a, b\rangle = \|a\|\|b\|\cos(\alpha)$, and (v) follows from the condition in the theorem. Substituting (28) back in (27), we get

$$\begin{aligned}
&\frac{\mathbb{E}\left[F(\boldsymbol{\theta}^{(t+1)})\right] - F(\boldsymbol{\theta}^{(t)})}{\eta\tau} \\
&\leq -\frac{1}{4}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + \frac{1}{2}\sum_{k=1}^{K}\omega_k \underbrace{\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t)}) - \boldsymbol{h}_k^{(t)}\right\|^2\right]}_{\mathbf{Z_4}} + 2\Delta^2 + \eta\beta\sigma^2,
\end{aligned} \tag{29}$$

where $\mathbf{Z_4}$ is given by,

$$\begin{aligned}
\mathbf{Z_4} &= \mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t)}) - \boldsymbol{h}_k^{(t)}\right\|^2\right] = \frac{1}{\tau^2}\mathbb{E}\left[\left\|\sum_{b=0}^{\tau-1}\left(\nabla F_k(\boldsymbol{\theta}_k^{(t,0)}) - \nabla F_k(\boldsymbol{\theta}_k^{(t,b)})\right)\right\|^2\right] \\
&\leq \frac{1}{\tau}\sum_{b=0}^{\tau-1}\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,0)}) - \nabla F_k(\boldsymbol{\theta}_k^{(t,b)})\right\|^2\right] \leq \frac{\beta^2}{\tau}\sum_{b=0}^{\tau-1}\mathbb{E}\left[\left\|\boldsymbol{\theta}_k^{(t,0)} - \boldsymbol{\theta}_k^{(t,b)}\right\|^2\right].
\end{aligned} \tag{30}$$

Also, using the local update rule $\boldsymbol{\theta}_k^{(t,b)} \leftarrow \boldsymbol{\theta}_k^{(t,0)} - \eta \sum_{s=0}^{b-1} \mathbf{g}_k(\boldsymbol{\theta}_k^{(t,s)})$, where $\boldsymbol{\theta}_k^{(t,b)}$ is the model parameter obtained at the b-th local iteration of the global round t at device k, we get:

$$\begin{aligned}
\mathbf{Z_5} &= \mathbb{E}\left[\left\|\boldsymbol{\theta}_k^{(t,0)} - \boldsymbol{\theta}_k^{(t,b)}\right\|^2\right] \\
&= \eta^2 \mathbb{E}\left[\left\|\sum_{s=0}^{b-1} \mathbf{g}_k(\boldsymbol{\theta}_k^{(t,s)})\right\|^2\right] \overset{(i)}{\leq} 2\eta^2 \mathbb{E}\left[\left\|\sum_{s=0}^{b-1} \nabla F_k(\boldsymbol{\theta}_k^{(t,s)})\right\|^2\right] + 2\eta^2 \mathbb{E}\left[\left\|\sum_{s=0}^{b-1} \boldsymbol{\epsilon}_k^{(t,s)}\right\|^2\right] \\
&\overset{(ii)}{\leq} 2\eta^2 b \sum_{s=0}^{b-1} \mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,s)})\right\|^2\right] + 2\eta^2 \sum_{s=0}^{b-1} \mathbb{E}\left[\left\|\boldsymbol{\epsilon}_k^{(t,s)}\right\|^2\right] \\
&\leq 2\eta^2 b \sum_{s=0}^{\tau-1} \mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,s)})\right\|^2\right] + 2\eta^2 \sigma^2 b,
\end{aligned} \tag{31}$$

where (i) uses Cauchy-Schwartz inequality and (ii) uses Lemma 1 and Cauchy-Schwartz inequality. Also note that

$$\sum_{b=0}^{\tau-1} b = \frac{\tau(\tau-1)}{2}. \tag{32}$$

Taking the cumulative sum of both hand sides of $\mathbf{Z_5}$ from (31) over all batches, i.e., $\frac{1}{\tau}\sum_{b=0}^{\tau-1}$, and using (32), we get:

$$\frac{1}{\tau}\sum_{b=0}^{\tau-1} \mathbb{E}\left[\left\|\boldsymbol{\theta}_k^{(t,0)} - \boldsymbol{\theta}_k^{(t,b)}\right\|^2\right] \leq \sigma^2\eta^2(\tau-1) + \eta^2(\tau-1)\sum_{b=0}^{\tau-1} \underbrace{\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,b)})\right\|^2\right]}_{\mathbf{Z_6}}. \tag{33}$$

Furthermore, term $\mathbf{Z_6}$ can be bounded as follows:

$$\begin{aligned}
\mathbf{Z_6} &= \mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,b)})\right\|^2\right] \overset{(i)}{\leq} 2\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,b)}) - \nabla F_k(\boldsymbol{\theta}_k^{(t,0)})\right\|^2\right] + 2\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,0)})\right\|^2\right] \\
&\overset{(ii)}{\leq} 2\beta^2 \mathbb{E}\left[\left\|\boldsymbol{\theta}_k^{(t,b)} - \boldsymbol{\theta}_k^{(t,0)}\right\|^2\right] + 2\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,0)})\right\|^2\right],
\end{aligned} \tag{34}$$

where (i) uses Cauchy-Schwartz inequality and (ii) uses (**A1**). Replacing $\mathbf{Z_6}$ in (33) using (34), we get:

$$\begin{aligned}
&\frac{1}{\tau}\sum_{b=0}^{\tau-1} \underbrace{\mathbb{E}\left[\left\|\boldsymbol{\theta}_k^{(t,0)} - \boldsymbol{\theta}_k^{(t,b)}\right\|^2\right]}_{\mathbf{Z_5}} \\
&\leq \eta^2\sigma^2(\tau-1) + 2\eta^2\beta^2(\tau-1)\sum_{b=0}^{\tau-1} \underbrace{\mathbb{E}\left[\left\|\boldsymbol{\theta}_k^{(t,b)} - \boldsymbol{\theta}_k^{(t,0)}\right\|^2\right]}_{\mathbf{Z_5}} + 2\eta^2\tau(\tau-1)\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,0)})\right\|^2\right].
\end{aligned} \tag{35}$$

Note that $\mathbf{Z_5}$, which is originally defined in (31), appears both in the left hand side (LHS) and right hand side (RHS) of the above expression. Rearranging the terms in the above inequality yields:

$$\frac{1}{\tau}\sum_{b=0}^{\tau-1} \mathbb{E}\left[\left\|\boldsymbol{\theta}_k^{(t,0)} - \boldsymbol{\theta}_k^{(t,b)}\right\|^2\right] \leq \frac{\eta^2\sigma^2(\tau-1)}{1-2\eta^2\beta^2\tau(\tau-1)} + \frac{2\eta^2\tau(\tau-1)}{1-2\eta^2\beta^2\tau(\tau-1)}\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,0)})\right\|^2\right]. \tag{36}$$

Defining $H = 2\eta^2\beta^2\tau(\tau-1)$, the above inequality can be re-written to evaluate $\mathbf{Z_4}$,

$$\begin{aligned}
\mathbf{Z_4} &= \frac{\beta^2}{\tau}\sum_{b=0}^{\tau-1}\mathbb{E}\left[\left\|\boldsymbol{\theta}^{(t,0)} - \boldsymbol{\theta}^{(t,b)}\right\|^2\right] \\
&\leq \frac{\eta^2\beta^2\sigma^2(\tau-1)}{1-2\eta^2\beta^2\tau(\tau-1)} + \frac{2\eta^2\beta^2\tau(\tau-1)}{1-2\eta^2\beta^2\tau(\tau-1)}\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,0)})\right\|^2\right] \\
&= \frac{\eta^2\beta^2\sigma^2}{1-H}(\tau-1) + \frac{H}{1-H}\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,0)})\right\|^2\right]. \qquad (37)
\end{aligned}$$

Taking a weighted sum from the both hand sides of the above inequality across all the workers and using (**A3**), we get:

$$\begin{aligned}
&\frac{1}{2}\sum_{k=1}^{K}\omega_k\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t)}) - \boldsymbol{h}_k^{(t)}\right\|^2\right] \\
&\leq \frac{\eta^2\beta^2\sigma^2(\tau-1)}{2(1-H)}\sum_{k=1}^{K}\omega_k + \frac{H}{2(1-H)}\sum_{k=1}^{K}\omega_k\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t,0)})\right\|^2\right] \\
&= \frac{\eta^2\beta^2\sigma^2(\tau-1)}{2(1-H)} + \frac{H}{2(1-H)}\sum_{k=1}^{K}\omega_k\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t)})\right\|^2\right] \\
&\overset{(i)}{\leq} \frac{\eta^2\beta^2\sigma^2(\tau-1)}{2(1-H)} + \frac{H\Upsilon^2}{2(1-H)}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2\right] + \frac{H\Gamma^2}{2(1-H)}, \qquad (38)
\end{aligned}$$

where (i) follows from (**A3**) and $\boldsymbol{\theta}_k^{(t)} = \boldsymbol{\theta}^{(t)}$, $\forall k$ since computation occurs at the instance of global aggregation. Next, plugging (38) back in (29), we get:

$$\begin{aligned}
&\frac{\mathbb{E}\left[F(\boldsymbol{\theta}^{(t+1)})\right] - F(\boldsymbol{\theta}^{(t)})}{\eta\tau} \\
&\leq -\frac{1}{4}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + \frac{1}{2}\sum_{k=1}^{K}\omega_k\mathbb{E}\left[\left\|\nabla F_k(\boldsymbol{\theta}_k^{(t)}) - \boldsymbol{h}_k^{(t)}\right\|^2\right] + 2\Delta^2 + \eta\beta\sigma^2 \\
&\leq -\frac{1}{4}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + \frac{\eta^2\beta^2\sigma^2(\tau-1)}{2(1-H)} + \frac{H\Upsilon^2}{2(1-H)}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2\right] + \frac{H\Gamma^2}{2(1-H)} \\
&\quad + 2\Delta^2 + \eta\beta\sigma^2 \\
&\leq -\frac{1}{4}\left(1 - \frac{2H\Upsilon^2}{1-H}\right)\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + \frac{\eta^2\beta^2\sigma^2(\tau-1)}{2(1-H)} + \frac{H\Gamma^2}{2(1-H)} + 2\Delta^2 + \eta\beta\sigma^2. \qquad (39)
\end{aligned}$$

If $H \leq \frac{1}{1+2\alpha\Upsilon^2}$ for some constant $\alpha > 1$, then it follows that $\frac{1}{1-H} \leq 1 + \frac{1}{2\alpha\Upsilon^2}$ and $\frac{2H\Upsilon^2}{1-H} \leq \frac{1}{\alpha}$. Choosing $\alpha = 2$ we can simplify the above expression as follows:

$$\begin{aligned}
&\frac{\mathbb{E}\left[F(\boldsymbol{\theta}^{(t+1)})\right] - F(\boldsymbol{\theta}^{(t)})}{\eta\tau} \\
&\leq -\frac{1}{8}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + 2\Delta^2 + \eta\beta\sigma^2 + \eta^2\beta^2\sigma^2(\tau-1)\left(\frac{1}{2} + \frac{1}{8\Upsilon^2}\right) + 2\eta^2\beta^2\Gamma^2\tau(\tau-1)\left(1 + \frac{1}{4\Upsilon^2}\right) \\
&\leq -\frac{1}{8}\left\|\nabla F(\boldsymbol{\theta}^{(t)})\right\|^2 + 2\Delta^2 + \eta\beta\sigma^2 + \frac{5}{8}\eta^2\beta^2\sigma^2(\tau-1) + \frac{5}{2}\eta^2\beta^2\Gamma^2\tau(\tau-1). \qquad (40)
\end{aligned}$$

Rearranging the terms in the above inequality and taking the average across all aggregation rounds from the both hand sides, yields:

$$
\begin{aligned}
&\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t,0)})\right\|^2\right] \\
&\leq \frac{8\left[\sum_{t=0}^{T-1}\mathbb{E}\left[F(\boldsymbol{\theta}^{(t)})\right]-F(\boldsymbol{\theta}^{(t+1)})\right]}{\eta\tau T}+16\Delta^2+8\eta\beta\sigma^2+5\eta^2\beta^2\sigma^2(\tau-1)+20\eta^2\beta^2\Gamma^2\tau(\tau-1) \\
&= \frac{8\left[F(\boldsymbol{\theta}^{(0)})-F(\boldsymbol{\theta}^{(T)})\right]}{\eta\tau T}+16\Delta^2+8\eta\beta\sigma^2+5\eta^2\beta^2\sigma^2(\tau-1)+20\eta^2\beta^2\Gamma^2\tau(\tau-1) \\
&\leq \frac{8\left[F(\boldsymbol{\theta}^{(0)})-F^\star\right]}{\eta\tau T}+16\Delta^2+8\eta\beta\sigma^2+5\eta^2\beta^2\sigma^2(\tau-1)+20\eta^2\beta^2\Gamma^2\tau(\tau-1),
\end{aligned}
\tag{41}
$$

where we used the fact that F is bounded below, since F_k-s are presumed to be bounded below, and $F^\star \leq F(\boldsymbol{\theta}), \forall \boldsymbol{\theta} \in \mathbb{R}^M$. This completes the proof of Theorem 1.

A.1 Condition on Learning Rate

From the two conditions on the learning rate used in the analysis above, we have

$$\eta\beta \leq \frac{1}{2\tau} \tag{42}$$

$$2\eta^2\beta^2\tau(\tau-1) \leq \frac{1}{1+4\Upsilon^2} \tag{43}$$

We can further tighten the second constraint as,

$$2\eta^2\beta^2\tau(\tau-1) \leq 2\eta^2\beta^2\tau^2 \leq \frac{1}{1+4\Upsilon^2} \tag{44}$$

Combining the two we have,

$$\eta\beta \leq \min\left\{\frac{1}{2\tau}, \frac{1}{\tau\sqrt{2(1+4\Upsilon^2)}}\right\}. \tag{45}$$

B PROOF OF COROLLARY 1

Using (41) we have:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t,0)})\right\|^2\right] \leq \frac{8\left[F(\boldsymbol{\theta}^{(0)})-F^\star\right]}{\eta\tau T} + 16\Delta^2 + 8\eta\beta\sigma^2 + 5\eta^2\beta^2\sigma^2(\tau-1) + 20\eta^2\beta^2\Gamma^2\tau(\tau-1). \tag{46}$$

Next, using the assumption in the corollary statement, we have $\Delta^2 \leq \eta$. We then can upper bound the RHS of (46) to get:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t,0)})\right\|^2\right] \leq \frac{8\left[F(\boldsymbol{\theta}^{(0)})-F^\star\right]}{\eta\tau T} + 16\eta + 8\eta\beta\sigma^2 + 5\eta^2\beta^2\sigma^2(\tau-1) + 20\eta^2\beta^2\Gamma^2\tau(\tau-1). \tag{47}$$

Choosing $\eta = \sqrt{\frac{1}{\tau T}}$, we get,

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t,0)})\right\|^2\right] \leq \frac{8\left[F(\boldsymbol{\theta}^{(0)})-F^\star\right]}{\sqrt{\tau T}} + \frac{8\beta\sigma^2}{\sqrt{\tau T}} + \frac{16}{\sqrt{\tau T}} + \frac{5\beta^2\sigma^2(\tau-1)}{\tau T} + \frac{20\beta^2\Gamma^2\tau(\tau-1)}{\tau T}. \tag{48}$$

We can write the above expression as,

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\left\|\nabla F(\boldsymbol{\theta}^{(t,0)})\right\|^2\right] \leq \mathcal{O}\left(\frac{1}{\sqrt{\tau T}}\right) + \mathcal{O}\left(\frac{\sigma^2}{\sqrt{\tau T}}\right) + \mathcal{O}\left(\frac{1}{\sqrt{\tau T}}\right) + \mathcal{O}\left(\frac{\sigma^2(\tau-1)}{\tau T})\right) + \mathcal{O}\left(\frac{(\tau-1)\Gamma^2}{T}\right). \tag{49}$$

This completes the proof for Corollary 1.

C ADDITIONAL PSEUDOCODES

In this Appendix, we provide a psuedocode for generating the preliminary experimental results in Sec. 2. The actual implementation of the following function calls used in the pseudocode can be found in the listed files of our code repository: https://github.com/PairML/LBGM:

- `get_num_PCA_components`: implemented in function estimate_optimal_ncomponents, file: src/common/nb_utils.py of the repository.
- `get_PCA_components`: implemented in function pca_transform, file: src/common/nb_utils.py of the repository.
- `cosine_similarity`: implemented using functions sklearn.preprocessing.normalize and numpy.dot, such that `cosine_similarity(a, b) = normalize(a).dot(normalize(b))` where a and b are numpy.array.
- `plot_1`, `plot_2`, and `plot_3`: implemented in files src/viz/prelim_1.py, src/viz/prelim_2.py, and src/viz/prelim_3.py respectively.

Algorithm 2 Pseudocode for Preliminary Experiments in Section 2

```
Initialize model parameter θ^(0).
Initialize actual_grads = { }                                   ▷ store gradients for PCA
Initialize pca95_store = { }                                    ▷ store #components accounting for 95% variance
Initialize pca99_store = { }                                    ▷ store #components accounting for 99% variance

for t = 0 to T − 1 do                                           ▷ training for T epochs
    Initialize g^(t) ← 0.                                       ▷ initialize gradient accumulator
    Set θ^(t,0) ← θ^(t).
    for b = 0 to B − 1 do                                       ▷ B minibatches per epoch
        Sample a minibatch of datapoints B from dataset D.
        Compute g^(t,b) = Σ_{d∈B} ∇f(θ^(t,b); d)/|B|.
        Update parameter: θ^(t,b+1) ← θ^(t,b) − η · g^(t,b).
        Accumulate gradient: g^(t) ← g^(t) + g^(t,b).
    end for
    Set θ^(t+1) ← θ^(t,B).
    actual_grads.append(g^(t))                                  ▷ append accumulated gradient to store
    pca95_store.append(get_num_PCA_components(actual_grads, variance = 0.95))
    pca99_store.append(get_num_PCA_components(actual_grads, variance = 0.99))

end for

plot_1(pca95_store, pca99_store)                                ▷ plot of PCA component progression

principal_grads = get_PCA_components(actual_grads, variance = 0.99)
heatmap = zeros(len(actual_grads), len(principal_grads))
for i = 0 to len(actual_grads) − 1 do
    for j = 0 to len(principal_grads) − 1 do
        heatmap[i, j] = cosine_similarity(actual_grads[i], principal_grads[j])
    end for
end for

plot_2(heatmap)                                                 ▷ plot of overlap of actual and principal gradients

heatmap = zeros(len(actual_grads), len(actual_grads))
for i = 0 to len(actual_grads) − 1 do
    for j = i to len(actual_grads) − 1 do
        heatmap[i, j] = heatmap[j, i] = cosine_similarity(actual_grads[i], actual_grads[j])
    end for
end for

plot_3(heatmap)                                                 ▷ plot of similarity among actual gradients
```

D ADDITIONAL PRELIMINARY EXPERIMENTS

We next present the additional experiments performed to test hypotheses, (**H1**) & (**H2**). As mentioned in Section 2, we study the rank-characteristics of centralized training using SGD on multiple datasets: FMNIST, MNIST, CIFAR-10, CelebA, COCO, and PascalVOC, and model architectures: CNN, FCN, Resnet18, VGG19, and U-Net. Section D.1 presents experiments complementary to the one presented in Fig. 1, while Section D.2 & D.3 present experiments complementary to those presented in Fig. 2, & 3, respectively. *Please follow the hyperlinks for the ease of navigating through the figures.*

D.1 PCA COMPONENT PROGRESSION

Together with Fig. 1, Figs. 10-13 show that both N99-PCA and N95-PCA are significantly lower than that the total number of gradients calculated during model training irrespective of model/dataset/learning task for multiple datasets and models which all agree with (**H1**). Specifically, the principal gradients (i.e., red and blue lines in the top row of the plots) are substantially lower (often as low as 10% of number of epochs, i.e., gradients generated) in these experiments. Refer below for the details of the figures.

1. Fig. 9 repeats the experiment conducted in Fig. 1 on CIFAR-100 using FCN, CNN, Resnet18, & VGG19.
2. Fig. 10 repeats the experiment conducted in Fig. 1 on MNIST using FCN, CNN, Resnet18, & VGG19.
3. Fig. 11 repeats the experiment conducted in Fig. 1 on FMNIST using FCN, CNN, Resnet18, & VGG19.
4. Fig. 12 repeats the experiment conducted in Fig. 1 on CIFAR-10, FMNIST, and MNIST using SVM, suggesting that we can use `LBGM` for classic classifiers that are not necessary neural networks.
5. Fig. 13 repeats the experiment conducted in Fig. 1 on COCO, and PascalVOC using U-Net.

D.2 OVERLAP OF ACTUAL AND PRINCIPAL GRADIENT

Next, we perform experiments summarized in Fig. 14-35 to further validate our observation in Fig. 2: (i) cosine similarity of actual gradients with principal gradients varies gradually over time, and (ii) actual gradients have a high cosine similarity with one or more of the principal gradients. Refer below for the details of the figures. *Each subplot is marked with #L, the layer number of the neural network, and #elem, the number of elements in each layer.*

1. Fig. 14 repeats the experiment conducted in Fig. 2 on CelebA using VGG19.
2. Fig. 15 repeats the experiment conducted in Fig. 2 on CelebA using Resnet18.
3. Fig. 16 repeats the experiment conducted in Fig. 2 on CelebA using FCN.
4. Fig. 17 repeats the experiment conducted in Fig. 2 on CelebA using CNN.
5. Fig. 18 repeats the experiment conducted in Fig. 2 on CIFAR-10 using VGG19.
6. Fig. 19 repeats the experiment conducted in Fig. 2 on CIFAR-10 using Resnet18.
7. Fig. 20 repeats the experiment conducted in Fig. 2 on CIFAR-10 using FCN.
8. Fig. 21 repeats the experiment conducted in Fig. 2 on CIFAR-10 using CNN.
9. Fig. 22 repeats the experiment conducted in Fig. 2 on CIFAR-100 using VGG19.
10. Fig. 23 repeats the experiment conducted in Fig. 2 on CIFAR-100 using Resnet18.
11. Fig. 24 repeats the experiment conducted in Fig. 2 on CIFAR-100 using FCN.
12. Fig. 25 repeats the experiment conducted in Fig. 2 on CIFAR-100 using CNN.
13. Fig. 26 repeats the experiment conducted in Fig. 2 on FMNIST using VGG19.
14. Fig. 27 repeats the experiment conducted in Fig. 2 on FMNIST using Resnet18.
15. Fig. 28 repeats the experiment conducted in Fig. 2 on FMNIST using FCN.

16. Fig. 29 repeats the experiment conducted in Fig. 2 on FMNIST using CNN.
17. Fig. 30 repeats the experiment conducted in Fig. 2 on MNIST using VGG19.
18. Fig. 31 repeats the experiment conducted in Fig. 2 on MNIST using Resnet18.
19. Fig. 32 repeats the experiment conducted in Fig. 2 on MNIST using FCN.
20. Fig. 33 repeats the experiment conducted in Fig. 2 on MNIST using CNN.
21. Fig. 34 repeats the experiment conducted in Fig. 2 on PascalVOC using U-Net.
22. Fig. 35 repeats the experiment conducted in Fig. 2 on COCO using U-Net.

D.3 Similarity among Consecutive Generated Gradients

Furthermore, we perform experiments summarized in Fig. 36-57. Together with Fig. 3, these experiments show that there is a significant overlap of consecutive gradients generated during SGD iterations, which further substantiates (**H2**) and bolsters our main idea that *"gradients transmitted in FL can be recycled/reused to represent the gradients generated in the subsequent iterations"*. Refer below for the details of the figures.

1. Fig. 36 repeats the experiment conducted in Fig. 3 on CelebA using VGG19.
2. Fig. 37 repeats the experiment conducted in Fig. 3 on CelebA using Resnet18.
3. Fig. 38 repeats the experiment conducted in Fig. 3 on CelebA using FCN.
4. Fig. 39 repeats the experiment conducted in Fig. 3 on CelebA using CNN.
5. Fig. 40 repeats the experiment conducted in Fig. 3 on CIFAR-10 using VGG19.
6. Fig. 41 repeats the experiment conducted in Fig. 3 on CIFAR-10 using Resnet18.
7. Fig. 42 repeats the experiment conducted in Fig. 3 on CIFAR-10 using FCN.
8. Fig. 43 repeats the experiment conducted in Fig. 3 on CIFAR-10 using CNN.
9. Fig. 44 repeats the experiment conducted in Fig. 3 on CIFAR-100 using VGG19.
10. Fig. 45 repeats the experiment conducted in Fig. 3 on CIFAR-100 using Resnet18.
11. Fig. 46 repeats the experiment conducted in Fig. 3 on CIFAR-100 using FCN.
12. Fig. 47 repeats the experiment conducted in Fig. 3 on CIFAR-100 using CNN.
13. Fig. 48 repeats the experiment conducted in Fig. 3 on FMNIST using VGG19.
14. Fig. 49 repeats the experiment conducted in Fig. 3 on FMNIST using Resnet18.
15. Fig. 50 repeats the experiment conducted in Fig. 3 on FMNIST using FCN.
16. Fig. 51 repeats the experiment conducted in Fig. 3 on FMNIST using CNN.
17. Fig. 52 repeats the experiment conducted in Fig. 3 on MNIST using VGG19.
18. Fig. 53 repeats the experiment conducted in Fig. 3 on MNIST using Resnet18.
19. Fig. 54 repeats the experiment conducted in Fig. 3 on MNIST using FCN.
20. Fig. 55 repeats the experiment conducted in Fig. 3 on MNIST using CNN.
21. Fig. 56 repeats the experiment conducted in Fig. 3 on PascalVOC using U-Net.
22. Fig. 57 repeats the experiment conducted in Fig. 3 on COCO using U-Net.

E Additional LBGM Experiments

In this section, we present complimentary experiments to the properties studies in Section 4 of the main text. In particular, we show that our observations hold for datasets: CIFAR-10, CIFAR-100, CelebA, FMNIST, and MNIST. We also present results when using shallower models FCN or deeper models Resnet18 different from CNNs. Section E.1 gives further evidence of utility of LBGM as a standalone solution. Section E.2 lists figures that summarize the effect of changing $\delta_k^{\text{threshold}}$ on model performance for mentioned datasets and models. In Section E.3 we list figures that summarize additional experiments to support the observations made in Fig. 7 and Section E.4 lists figures for

additional experiments corresponding to the observations made in Fig. 8. *Since the two layer FCN considered is a simple classifier, it does not perform well on complex datasets such as CIFAR-10, CIFAR-100, and CelebA. Thus, the respective results are omitted for FCN on these complicated datasets and the performance of FCN is only studied for MNIST and FMNIST datasets. Similarly, the 4-layer CNN architecture does not perform well on CIFAR-100 dataset and hence the corresponding results are ommited.*

E.1 LBGM AS A STANDALONE ALGORITHM.

1. Fig. 58 shows the result of repeating the experiment conducted in Fig. 5 (CNNs on non-iid data distribution) on CNNs for iid data distribution for datasets CIFAR-10, FMNIST, and MNIST.
2. Fig. 59 shows the result of repeating the experiment conducted in Fig. 5 (CNNs on non-iid data distribution) on FCNs for both non-iid and iid data distributions on datasets FMNIST and MNIST.
3. Fig. 60 shows the result of repeating the experiment conducted in Fig. 5 (CNNs on non-iid data distribution) on Resnet18s for non-iid data distribution on datasets CelebA, CIFAR-10, CIFAR-100, and MNIST using a setup similar to that of Wang et al. (2018).

E.2 EFFECT OF $\delta_k^{\text{threshold}}$ ON LBGM.

1. Fig. 61 shows the result of repeating the experiment conducted in Fig. 6 (CNNs on non-iid data distribution) on CNNs for iid data distribution for datasets CIFAR-10, FMNIST, and MNIST.
2. Fig. 62 shows the result of repeating the experiment conducted in Fig. 6 (CNNs on non-iid data distribution) on FCNs for both non-iid and iid data distributions on datasets FMNIST and MNIST.
3. Fig. 63 shows the result of repeating the experiment conducted in Fig. 6 (CNNs on non-iid data distribution) on Resnet18s for non-iid data distribution on datasets CelebA, CIFAR-10 and CIFAR-100 using a setup similar to that of Wang et al. (2018).

E.3 LBGM AS A PLUG-AND-PLAY ALGORITHM.

1. Fig. 64 shows the result of repeating the experiment conducted in Fig. 7 (CNNs on non-iid data distribution) on CNNs for iid data distribution for datasets CIFAR-10, FMNIST, and MNIST.
2. Fig. 65 shows the result of repeating the experiment conducted in Fig. 7 (CNNs on non-iid data distribution) on FCNs for both non-iid and iid data distributions on datasets FMNIST and MNIST.
3. Fig. 66 shows the result of repeating the experiment conducted in Fig. 7 (CNNs on non-iid data distribution) on Resnet18s for non-iid data distribution on datasets CelebA, CIFAR-10 and CIFAR-100 using a setup similar to that of Wang et al. (2018).

E.4 GENERALIZABILITY OF LBGM TO DISTRIBUTED TRAINING.

1. Fig. 67 shows the result of repeating the experiment conducted in Fig. 8 (CNNs on non-iid data distribution) on CNNs for iid data distribution for datasets CIFAR-10, FMNIST, and MNIST.
2. Fig. 68 shows the result of repeating the experiment conducted in Fig. 8 (CNNs on non-iid data distribution) on FCNs for both non-iid and iid data distributions on datasets FMNIST and MNIST.
3. Fig. 69 shows the result of repeating the experiment conducted in Fig. 8 (CNNs on non-iid data distribution) on Resnet18s for non-iid data distribution on datasets CelebA, CIFAR-10 and CIFAR-100 using a setup similar to that of Wang et al. (2018).

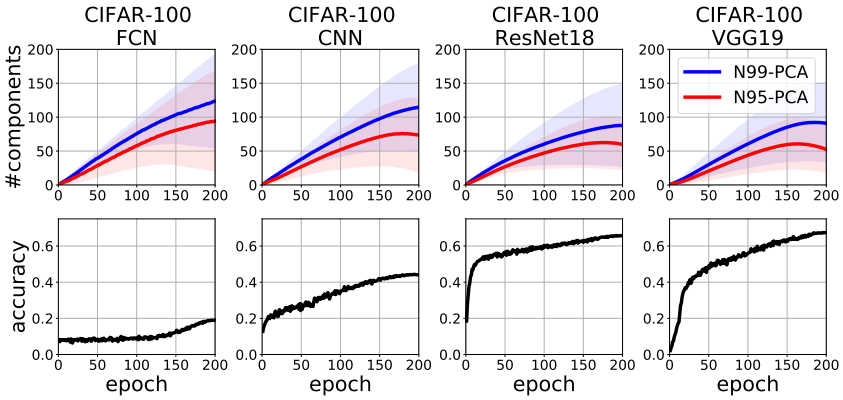

Figure 9: *PCA Components Progression*. Repeat of Fig. 1 on CIFAR-100 dataset.

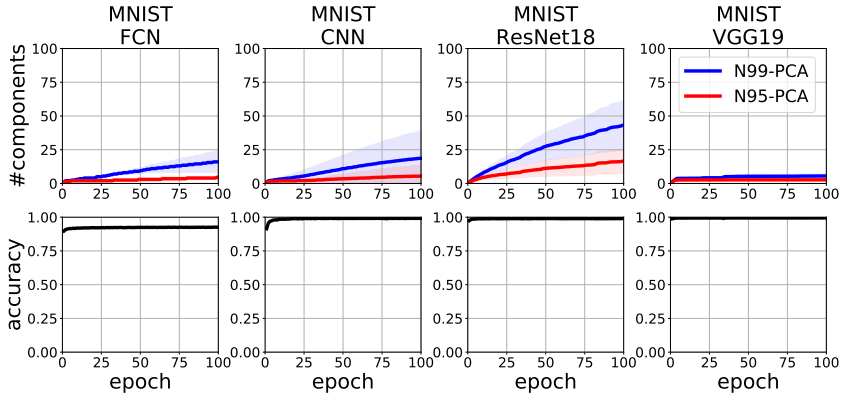

Figure 10: *PCA Components Progression*. Repeat of Fig. 1 on MNIST dataset.

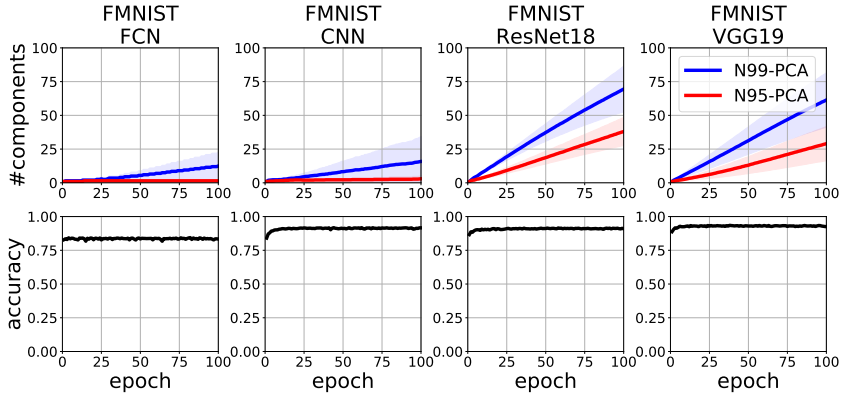

Figure 11: *PCA Components Progression*. Repeat of Fig. 1 on FMNIST dataset.

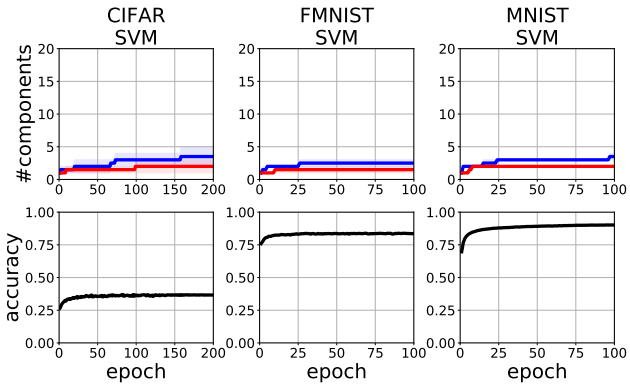

Figure 12: *PCA Components Progression*. Repeat of Fig. 1 on CIFAR-10, F-MNIST, and MNIST datasets but using squared SVM classifier.

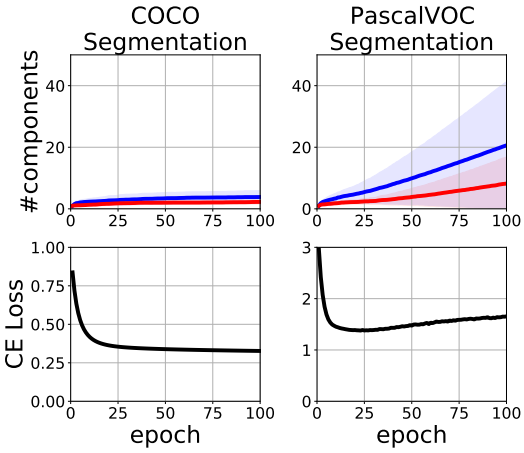

Figure 13: *PCA Components Progression*. Repeat of Fig. 1 on COCO, and PascalVOC datasets but using U-Net classifier.

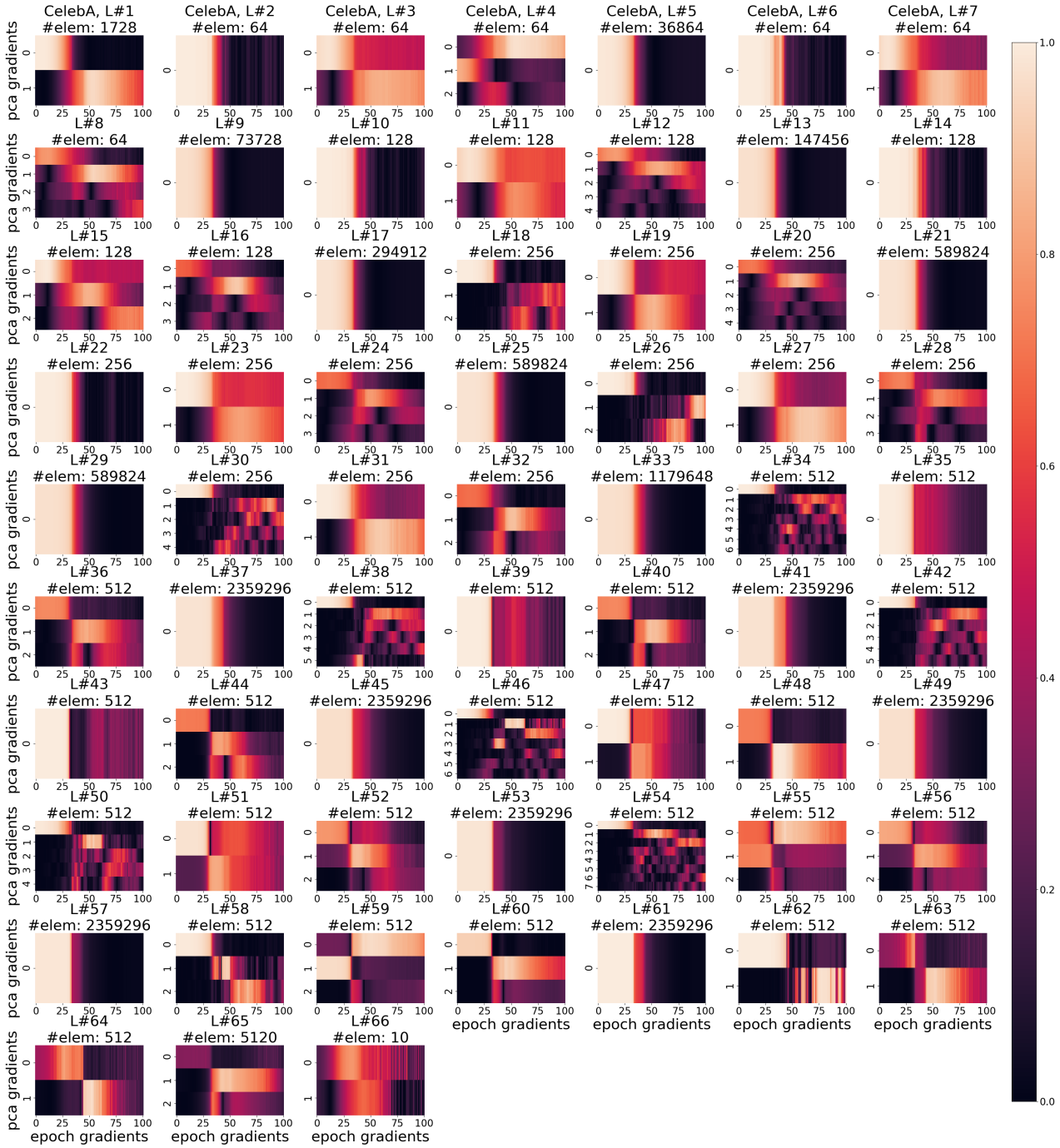

Figure 14: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **VGG19** trained on **CelebA** dataset.

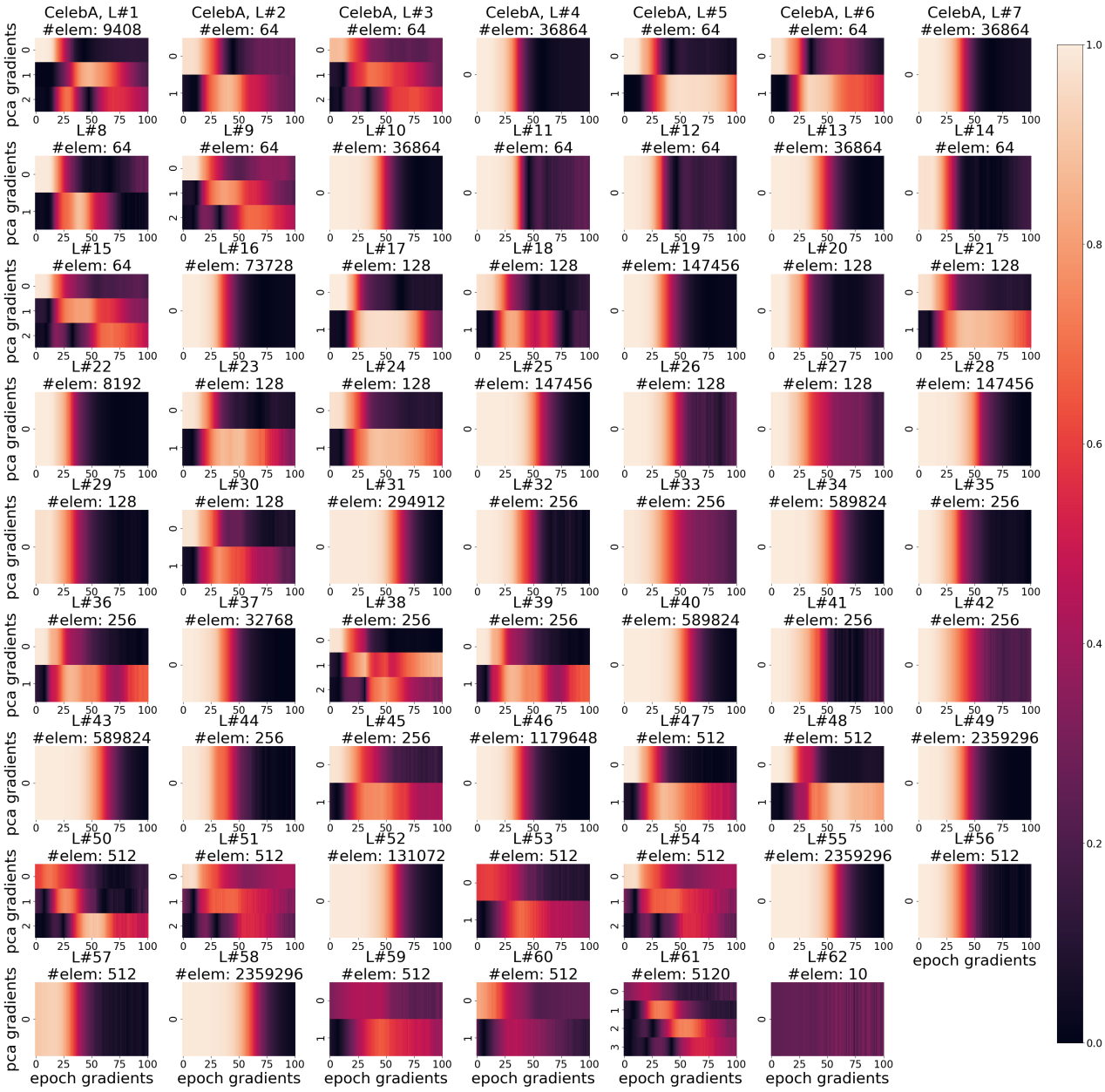

Figure 15: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **ResNet18** trained on **CelebA** dataset.

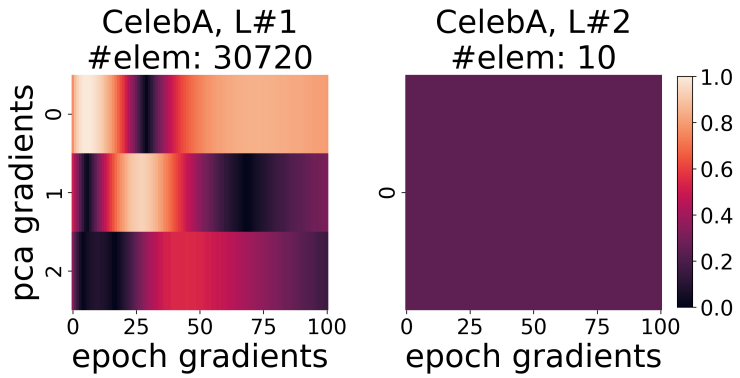

Figure 16: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **FCN** trained on **CelebA** dataset.

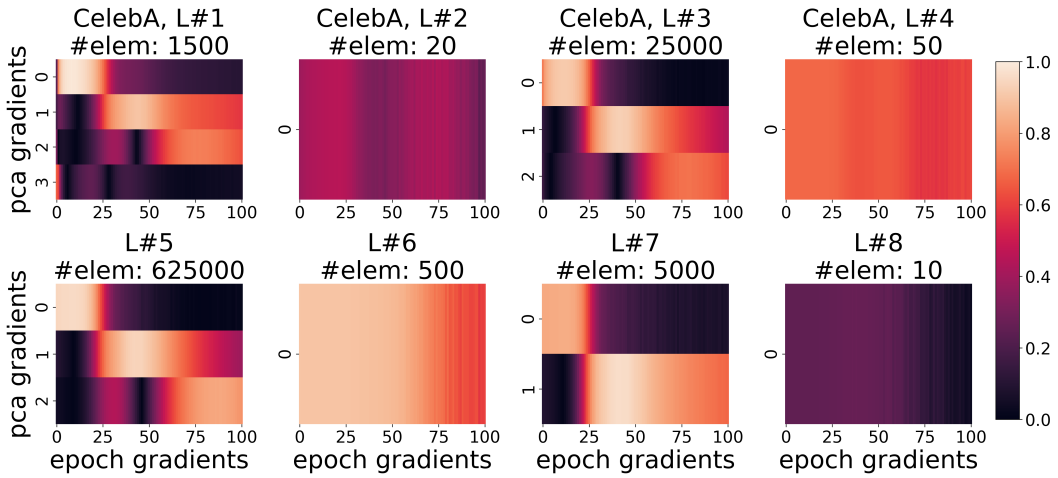

Figure 17: *PCA Components Overlap with Gradient*. Fig. 2 on **CNN** trained on **CelebA** dataset.

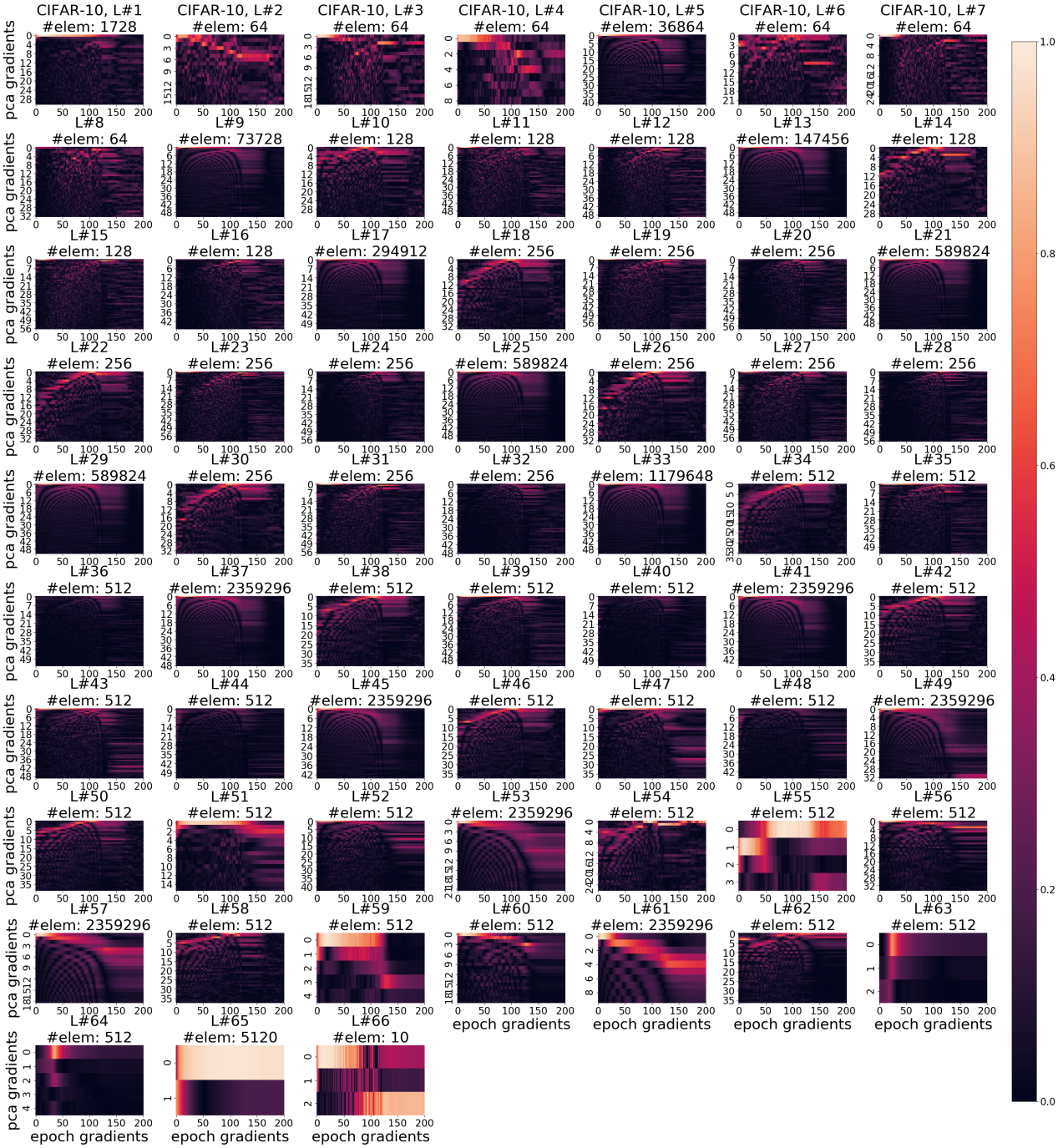

Figure 18: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **VGG19** trained on **CIFAR-10** dataset.

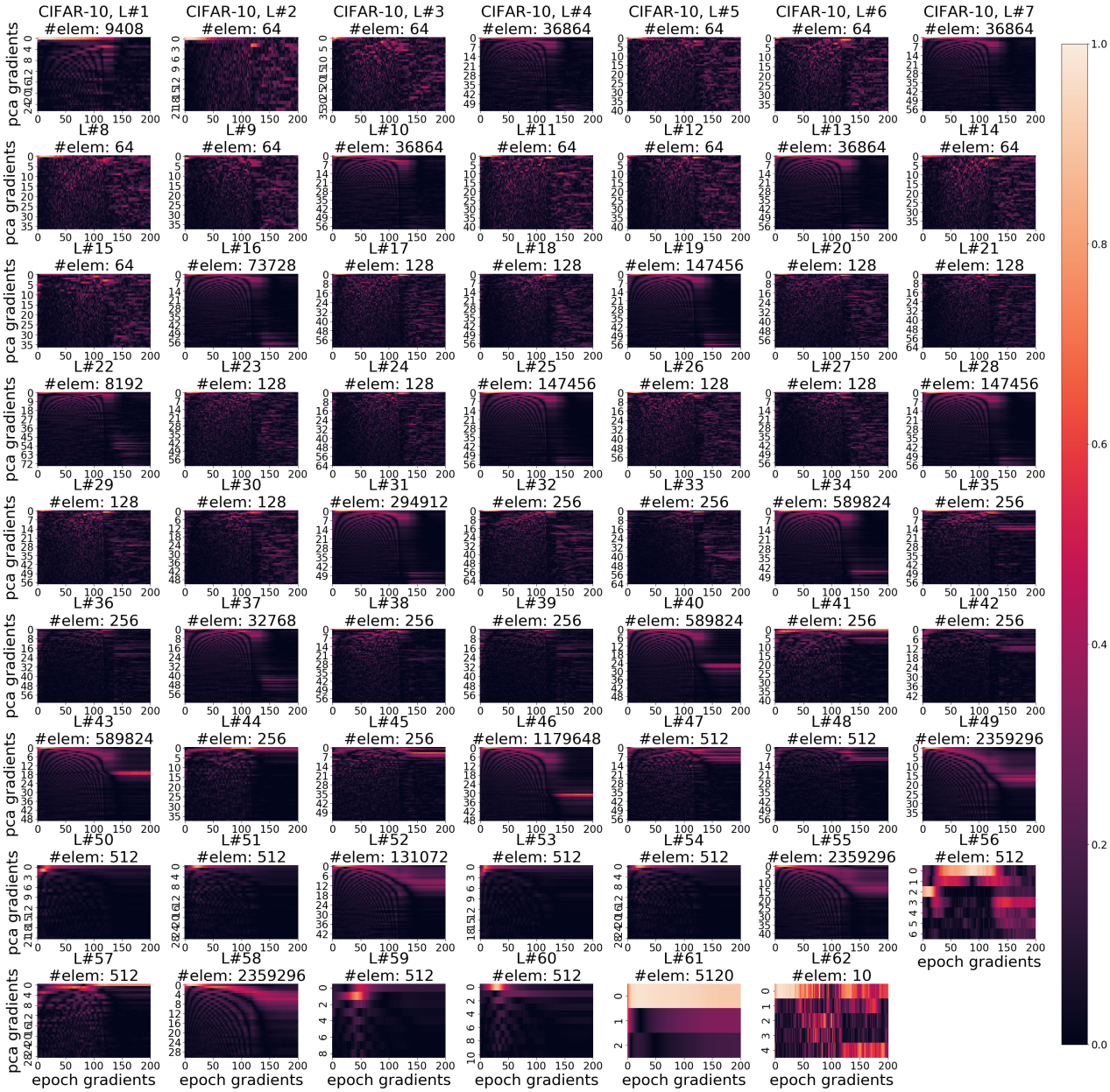

Figure 19: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **ResNet18** trained on **CIFAR-10** dataset.

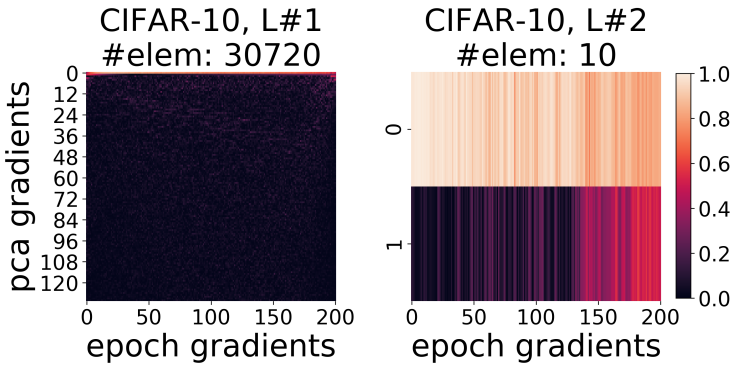

Figure 20: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **FCN** trained on **CIFAR-10** dataset.

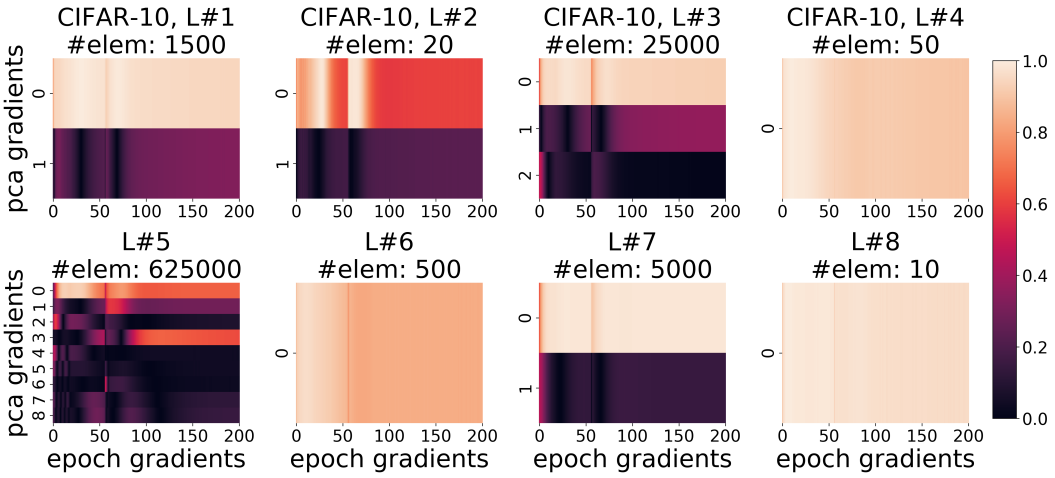

Figure 21: *PCA Components Overlap with Gradient*. Fig. 2 on **CNN** trained on **CIFAR-10** dataset.

Figure 22: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **VGG19** trained on **CIFAR-100** dataset.

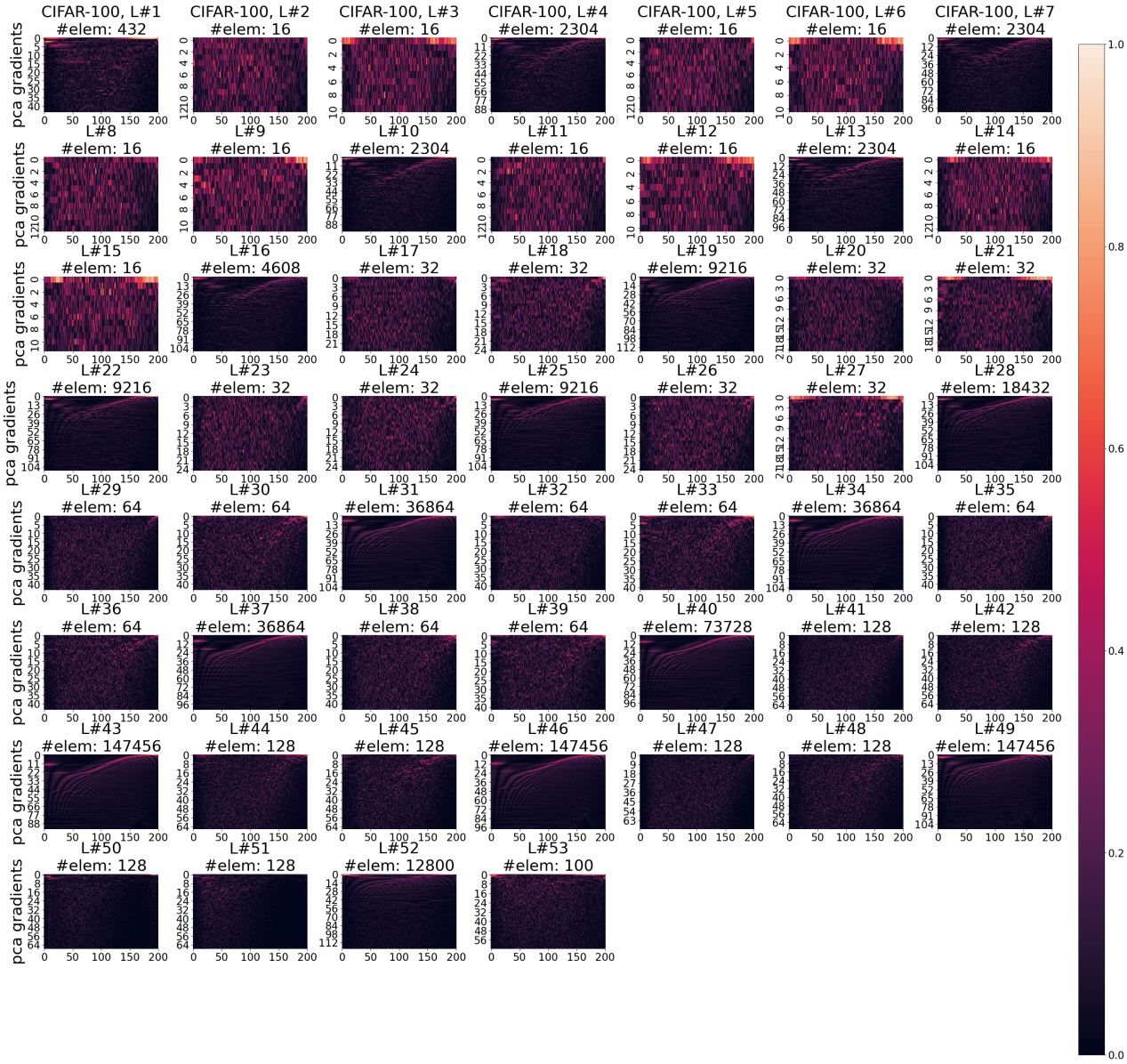

Figure 23: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **ResNet18** trained on **CIFAR-100** dataset.

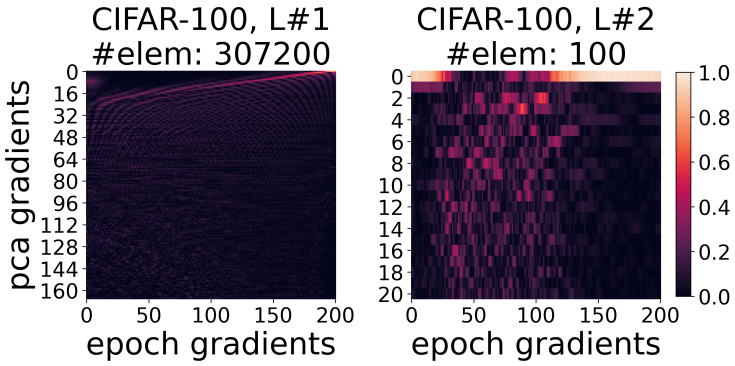

Figure 24: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **FCN** trained on **CIFAR-100** dataset.

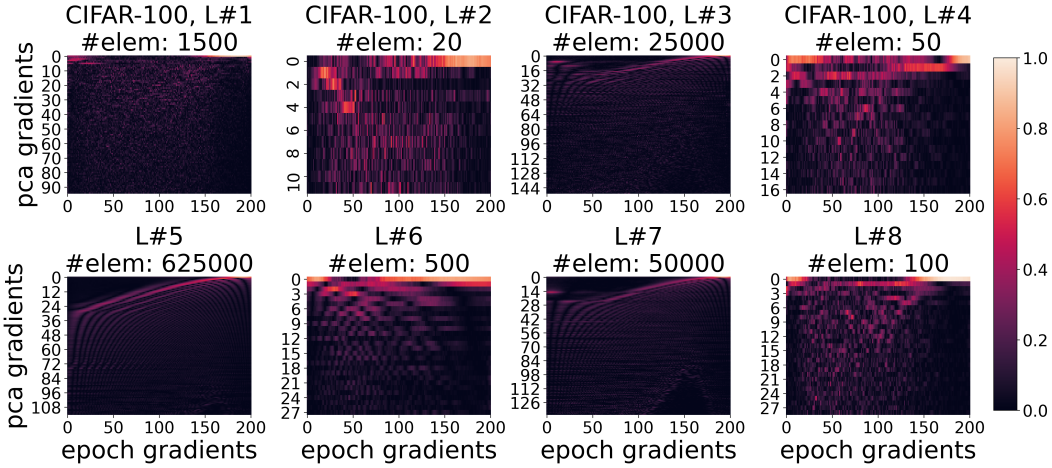

Figure 25: *PCA Components Overlap with Gradient*. Fig. 2 on **CNN** trained on **CIFAR-100** dataset.

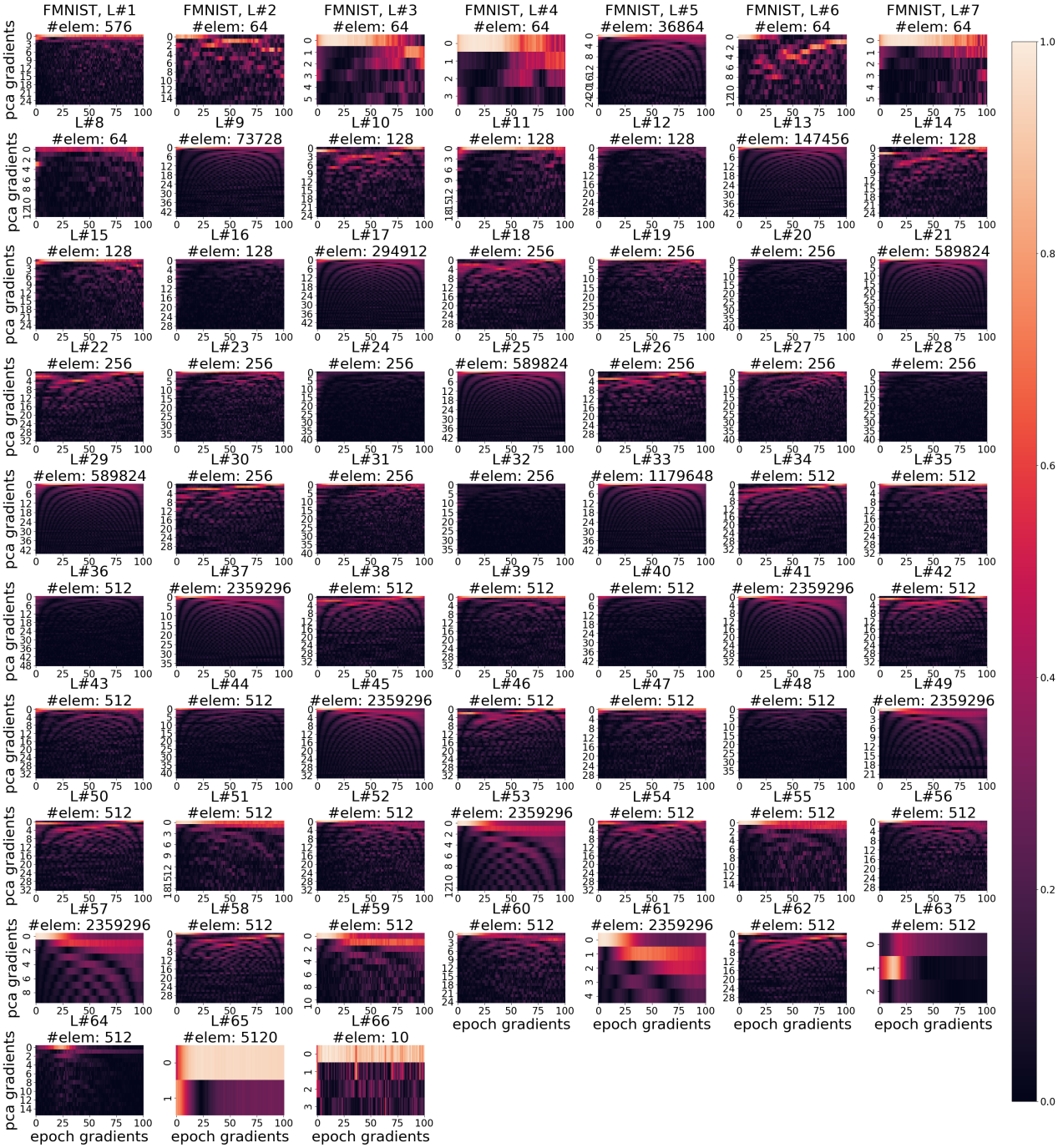

Figure 26: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **VGG19** trained on **FMNIST** dataset.

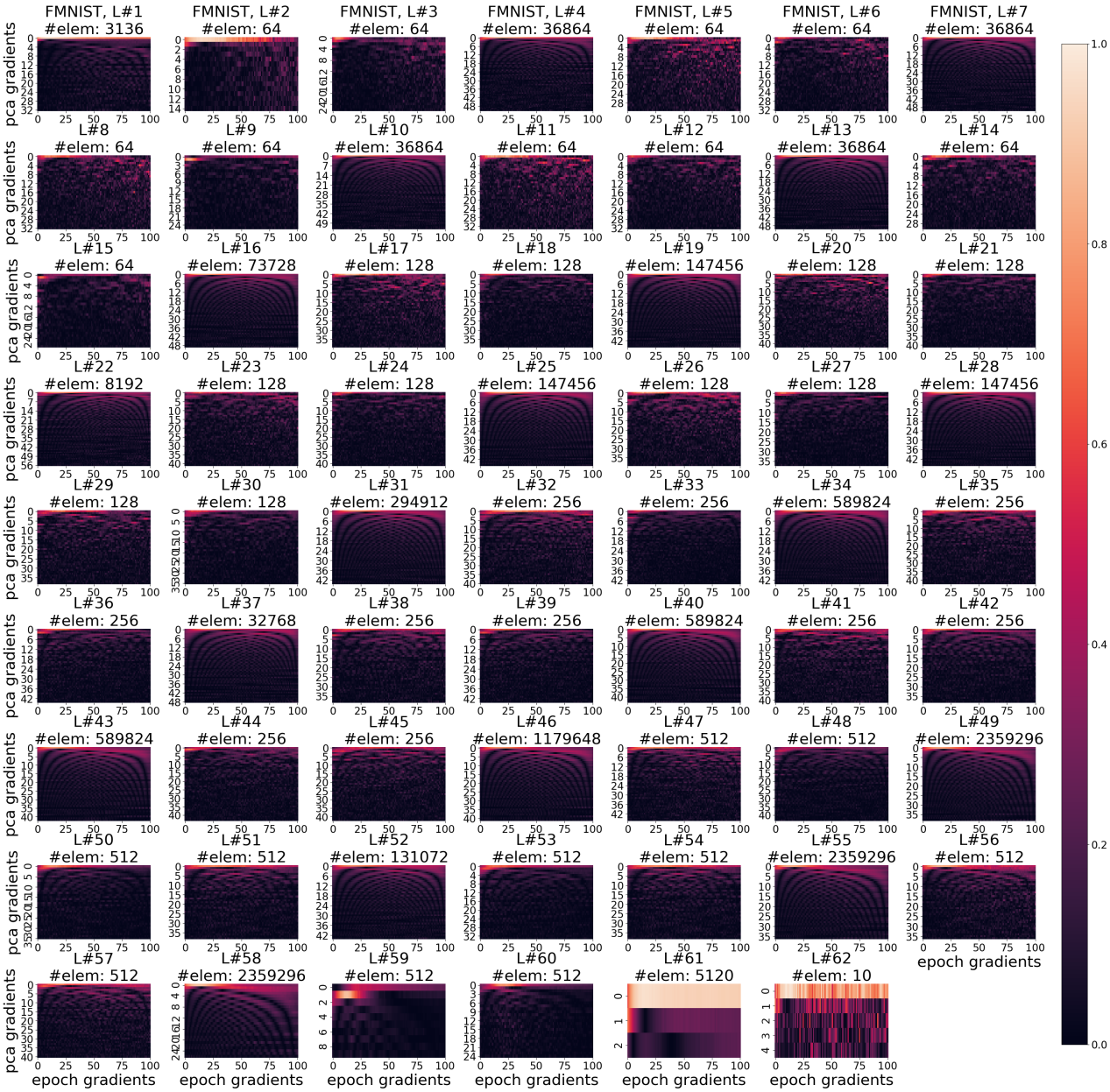

Figure 27: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **ResNet18** trained on **FMNIST** dataset.

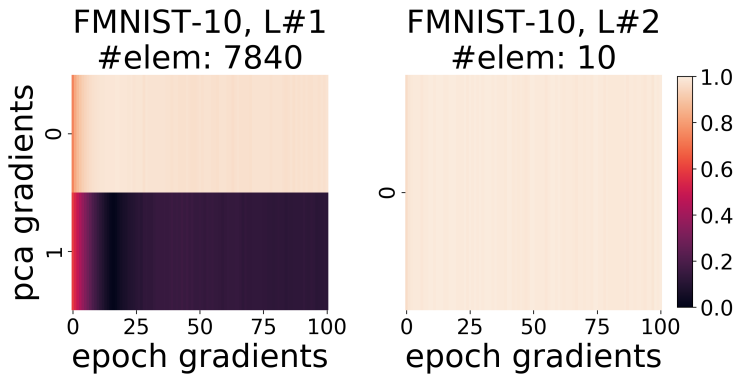

Figure 28: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **FCN** trained on **FMNIST** dataset.

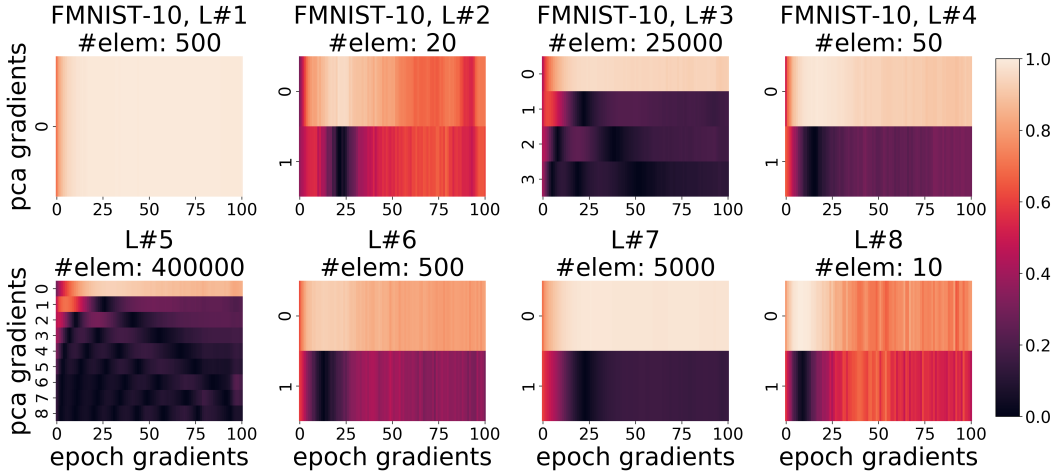

Figure 29: *PCA Components Overlap with Gradient*. Fig. 2 on **CNN** trained on **FMNIST** dataset.

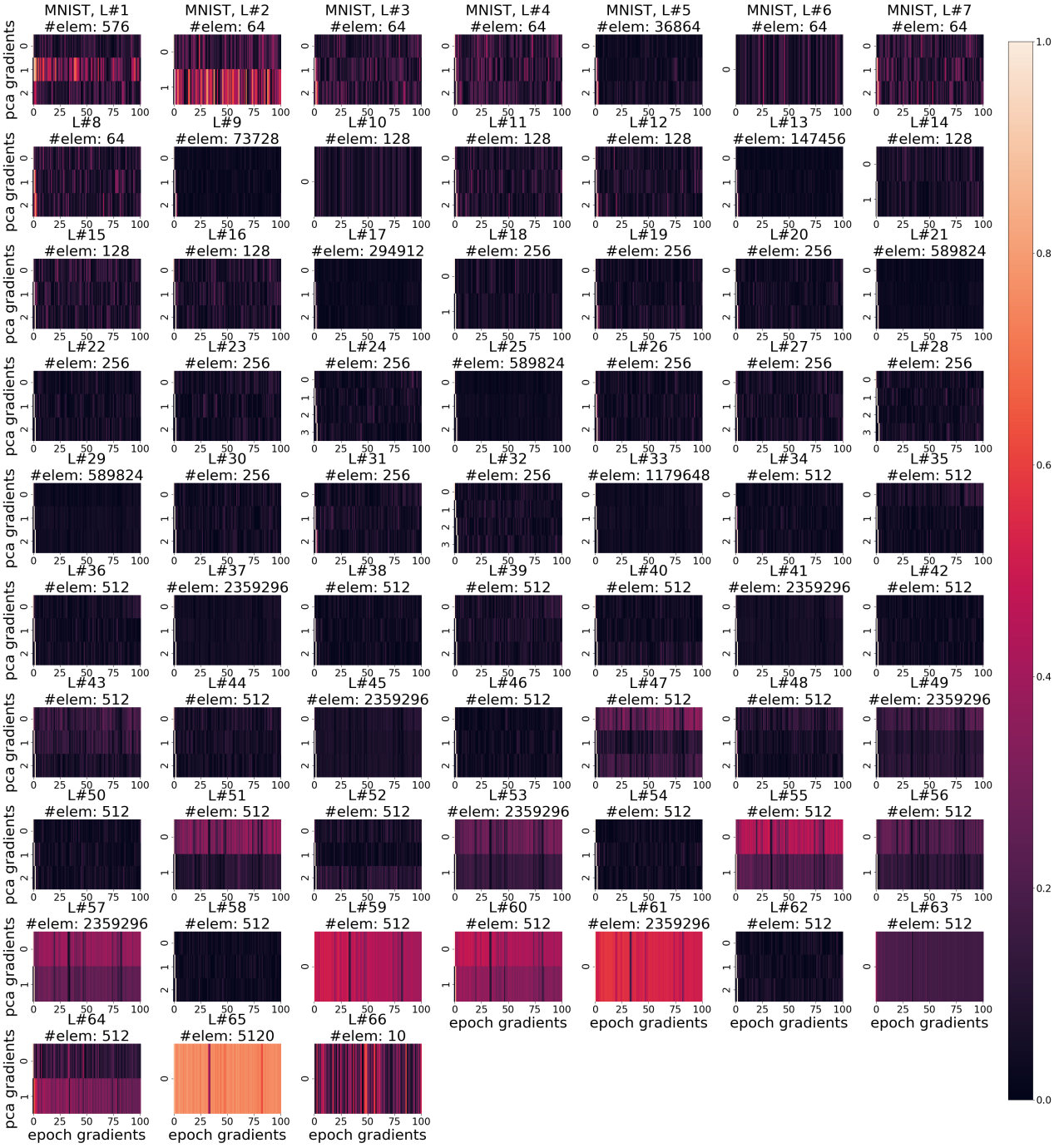

Figure 30: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **VGG19** trained on **MNIST** dataset.

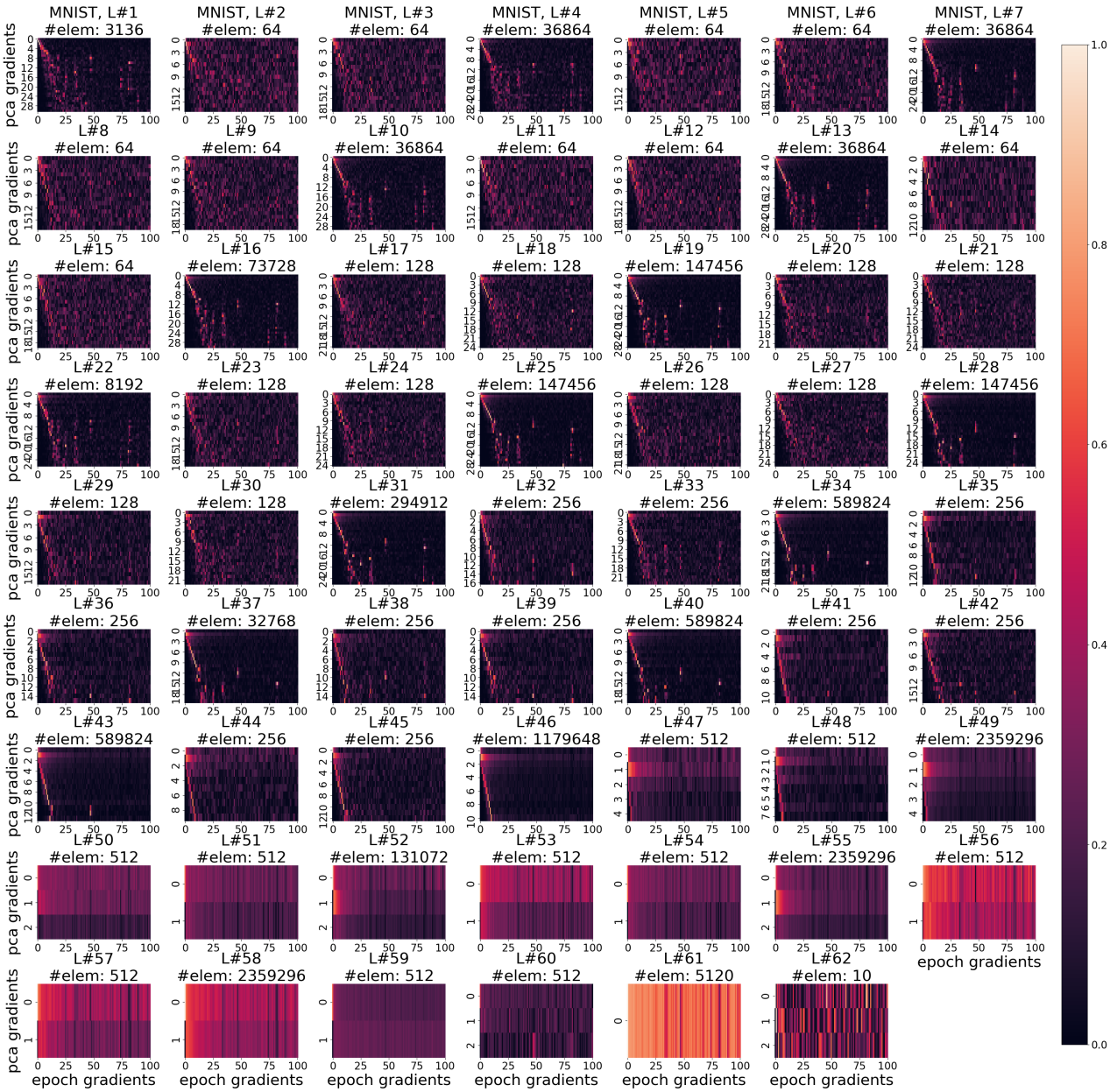

Figure 31: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **ResNet18** trained on **MNIST** dataset.

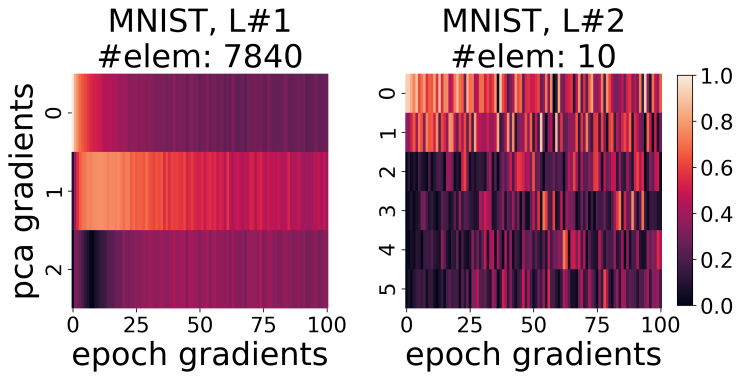

Figure 32: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **FCN** trained on **MNIST** dataset.

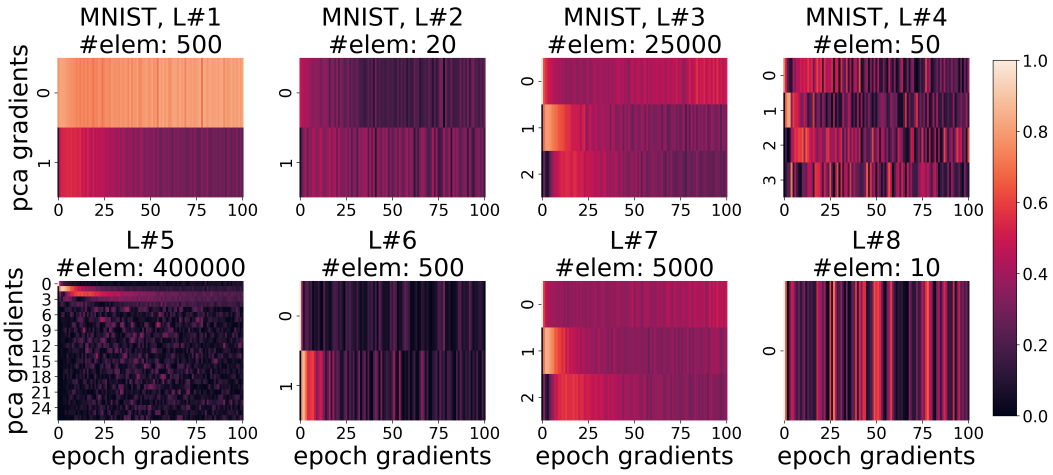

Figure 33: *PCA Components Overlap with Gradient*. Fig. 2 on **CNN** trained on **MNIST** dataset.

Figure 34: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **U-Net** trained on **PascalVOC** dataset.

Figure 35: *PCA Components Overlap with Gradient*. Repeat of Fig. 2 on **U-Net** trained on **COCO** dataset.

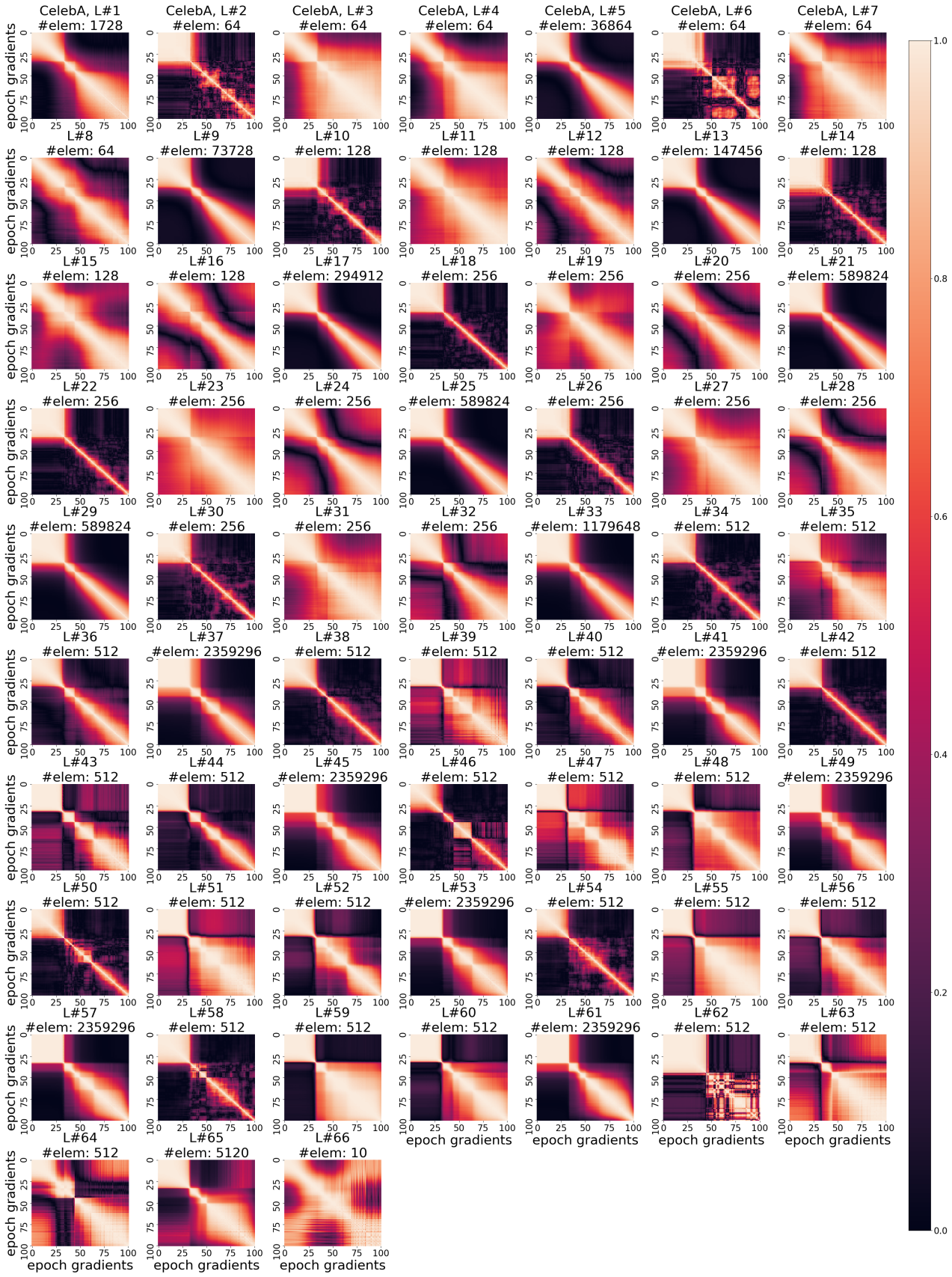

Figure 36: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **VGG19** trained on **CelebA** dataset.

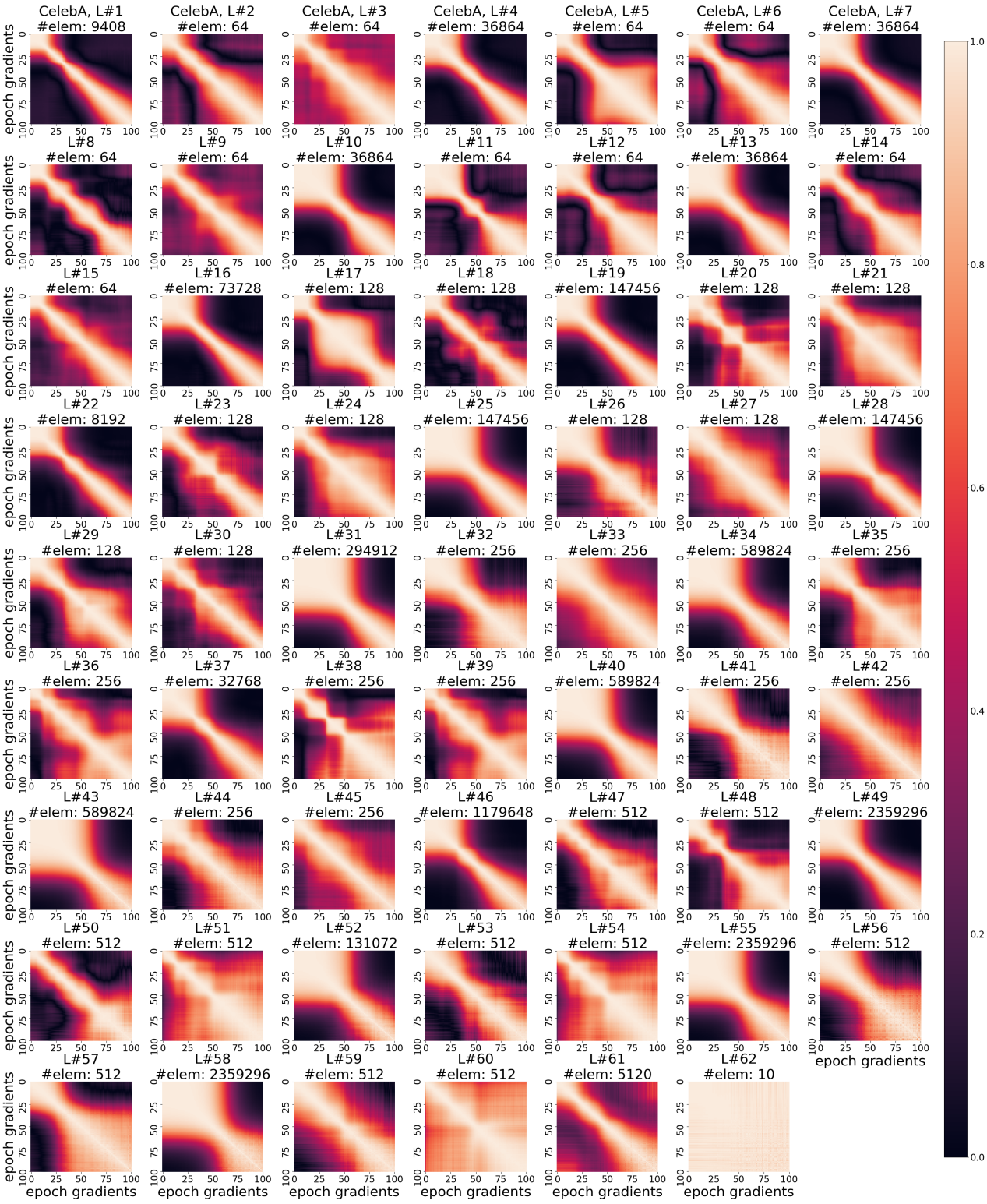

Figure 37: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **ResNet18** trained on **CelebA** dataset.

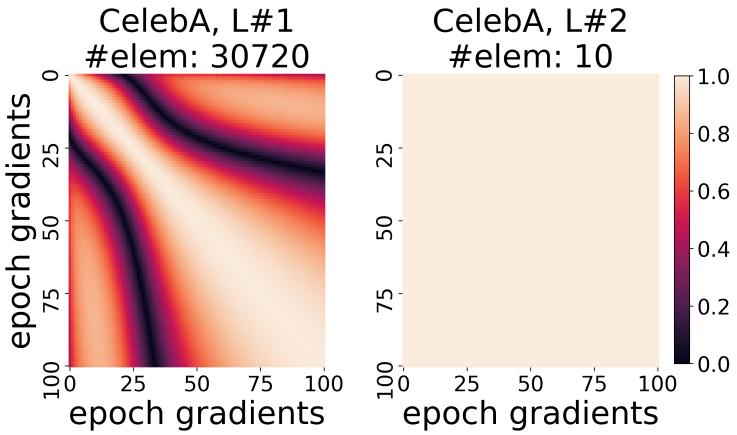

Figure 38: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **FCN** trained on **CelebA** dataset.

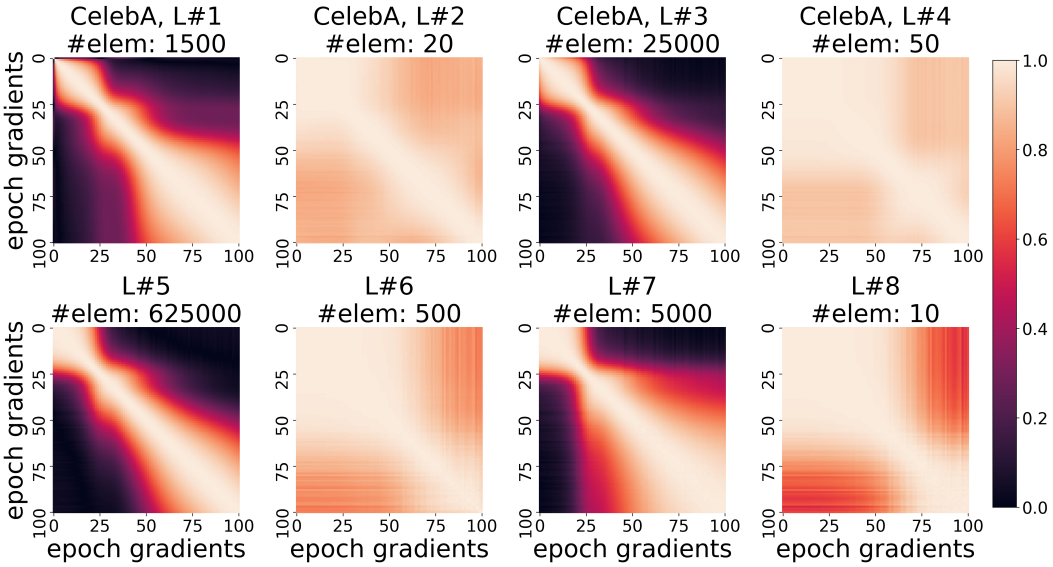

Figure 39: *PCA Components Overlap with Gradient*. Fig. 3 on **CNN** trained on **CelebA** dataset.

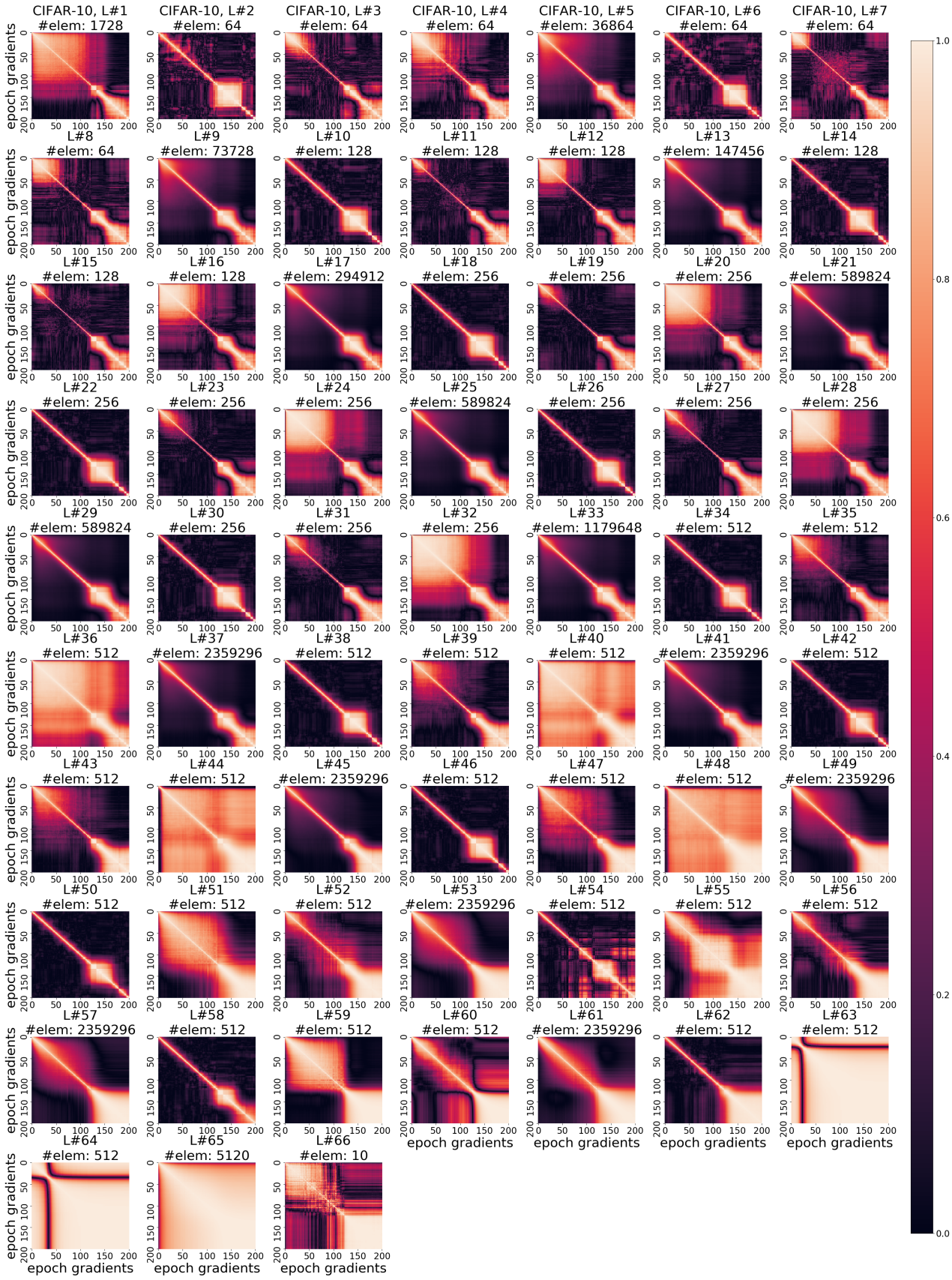

Figure 40: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **VGG19** trained on **CIFAR-10** dataset.

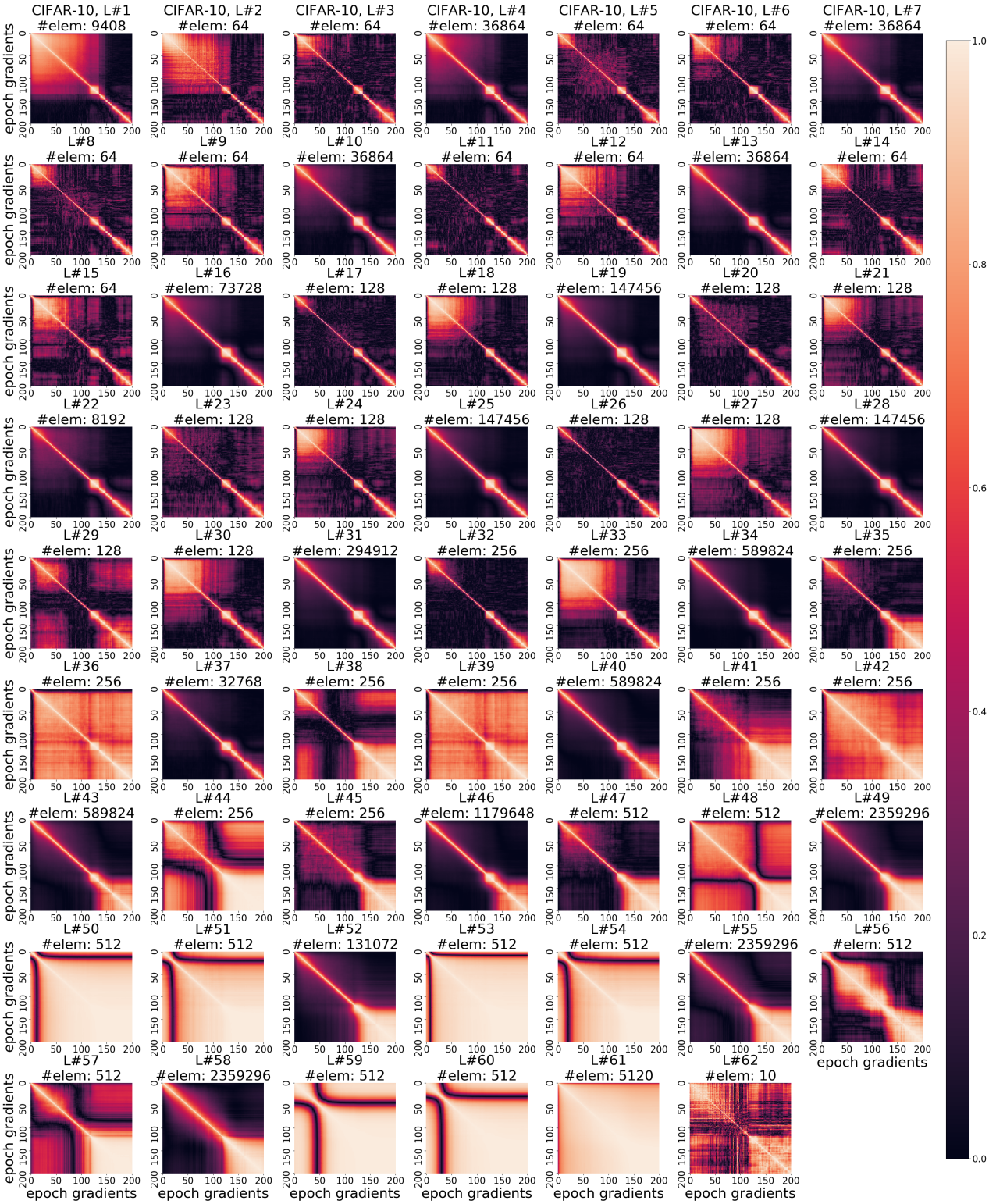

Figure 41: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **ResNet18** trained on **CIFAR-10** dataset.

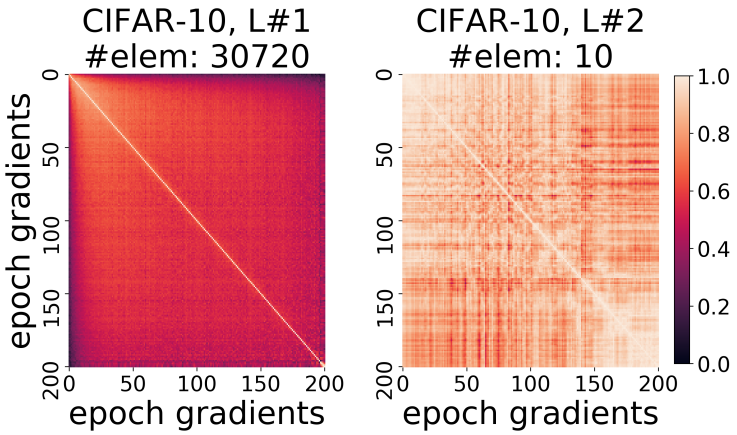

Figure 42: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **FCN** trained on **CIFAR-10** dataset.

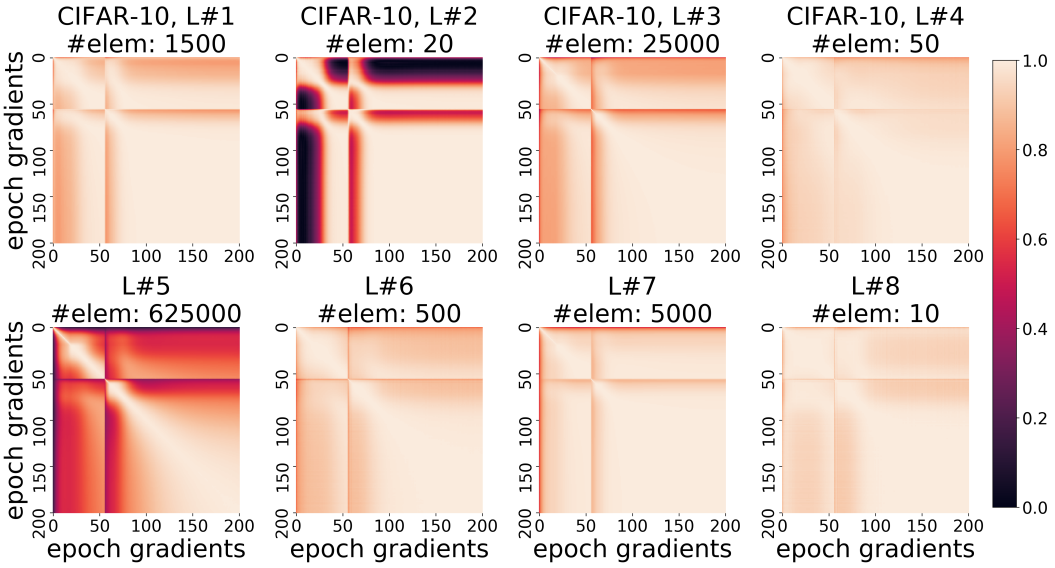

Figure 43: *PCA Components Overlap with Gradient*. Fig. 3 on **CNN** trained on **CIFAR-10** dataset.

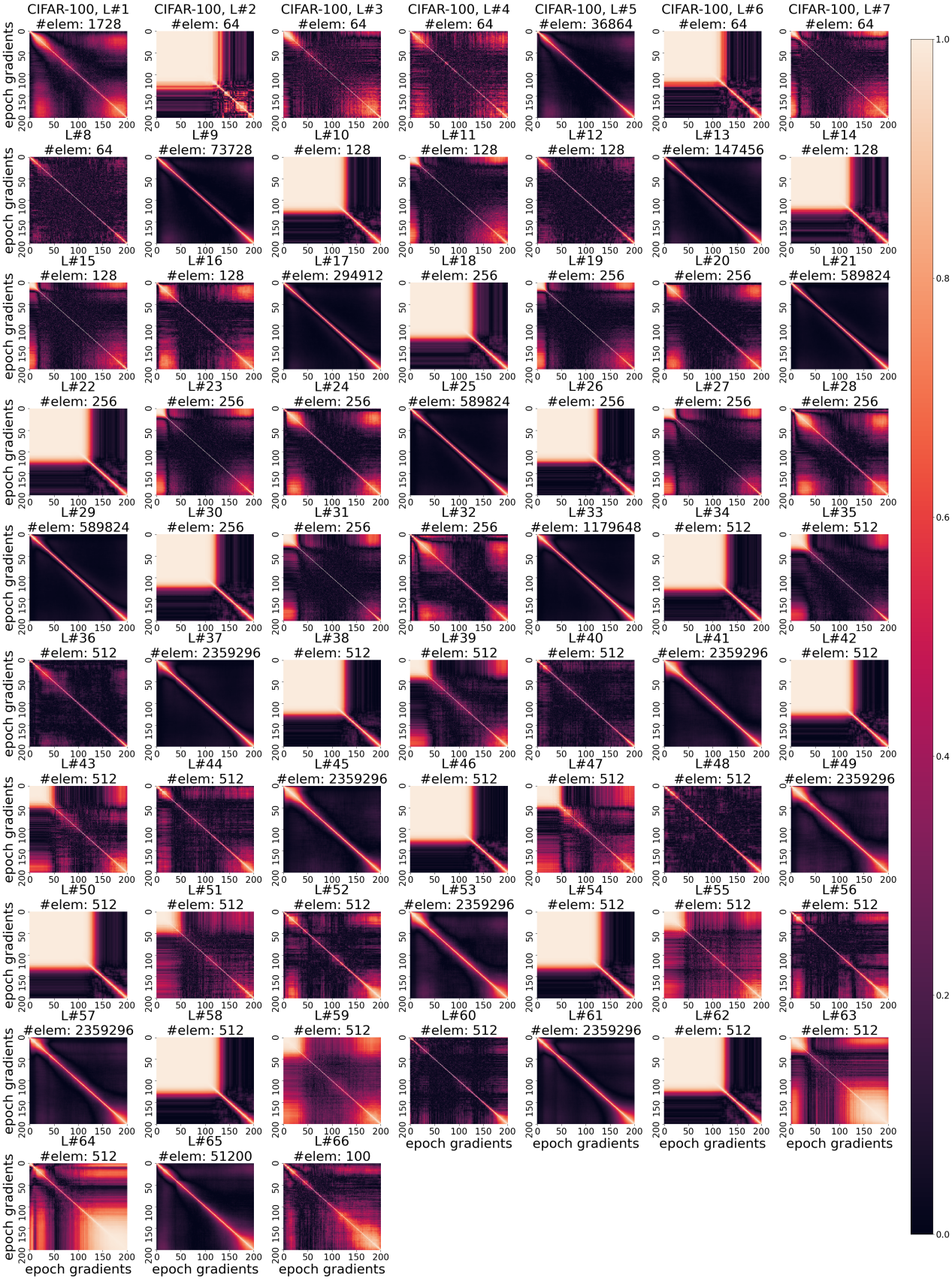

Figure 44: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **VGG19** trained on **CIFAR-100** dataset.

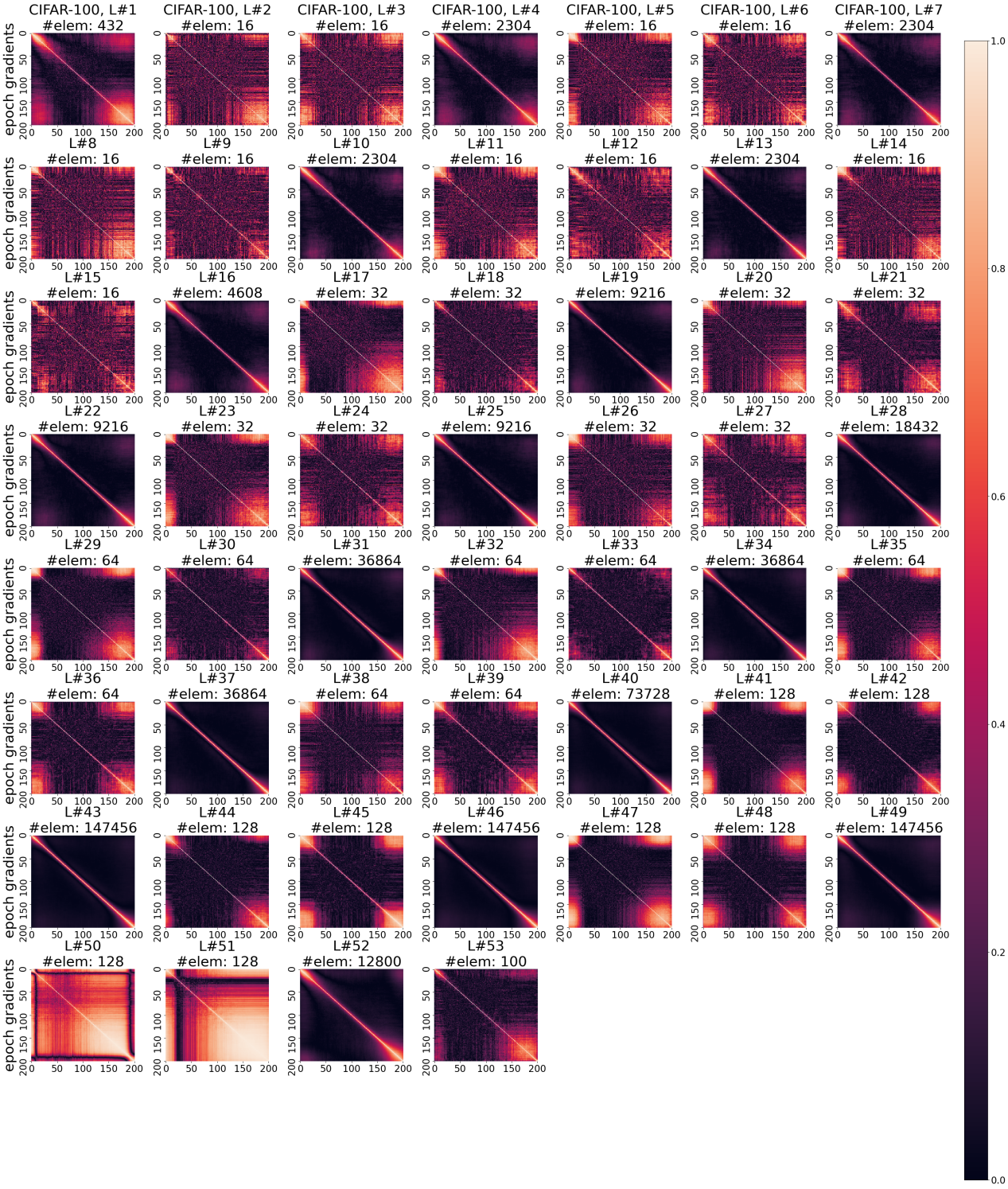

Figure 45: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **ResNet18** trained on **CIFAR-100** dataset.

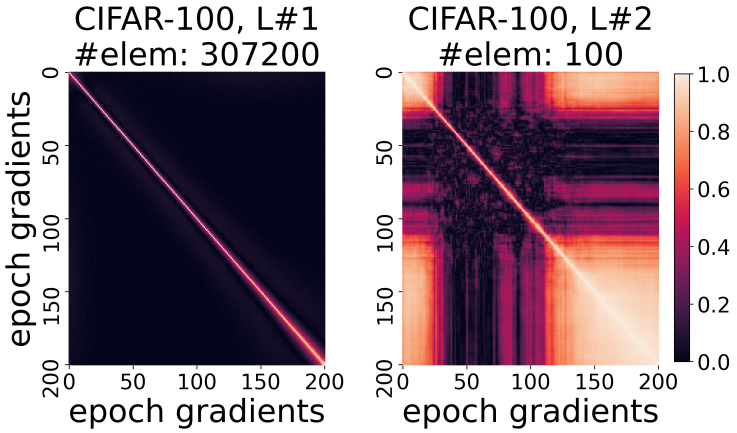

Figure 46: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **FCN** trained on **CIFAR-100** dataset.

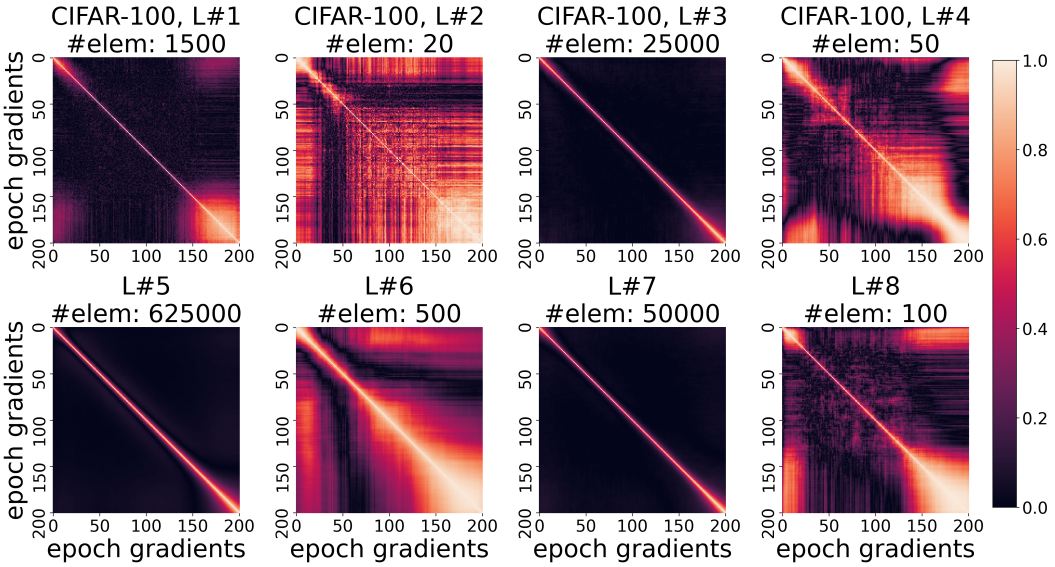

Figure 47: *PCA Components Overlap with Gradient*. Fig. 3 on **CNN** trained on **CIFAR-100** dataset.

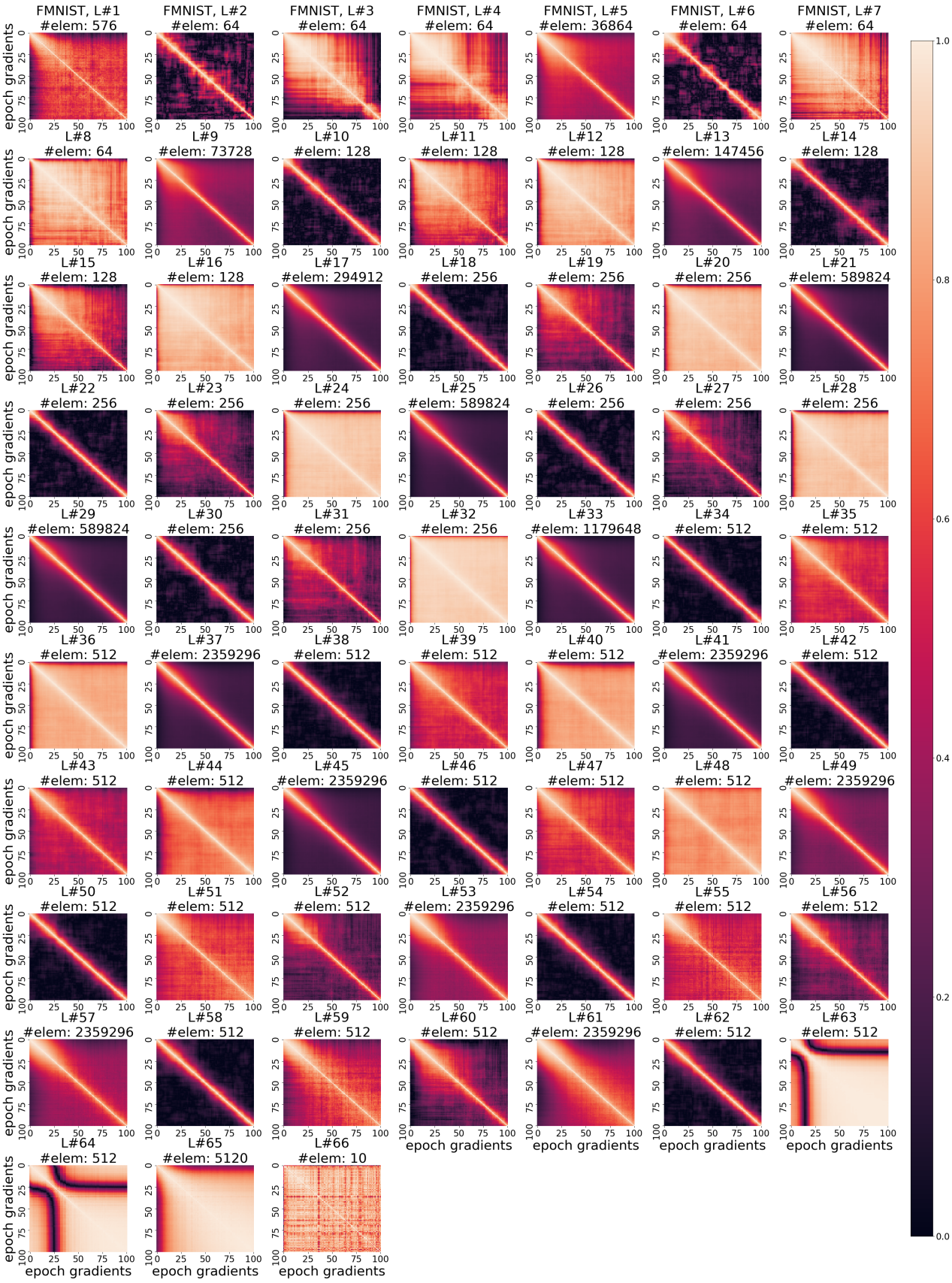

Figure 48: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **VGG19** trained on **FMNIST** dataset.

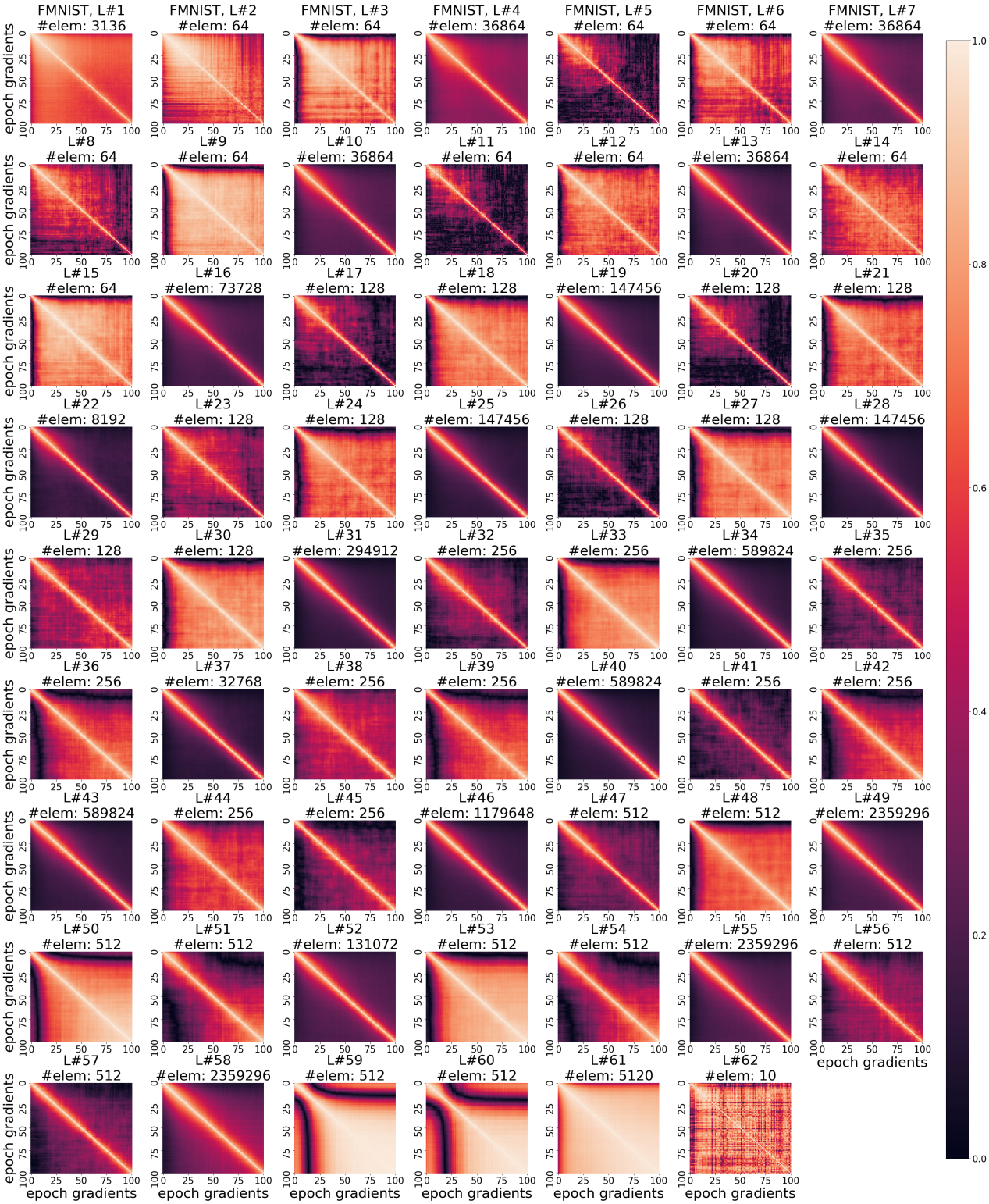

Figure 49: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **ResNet18** trained on **FMNIST** dataset.

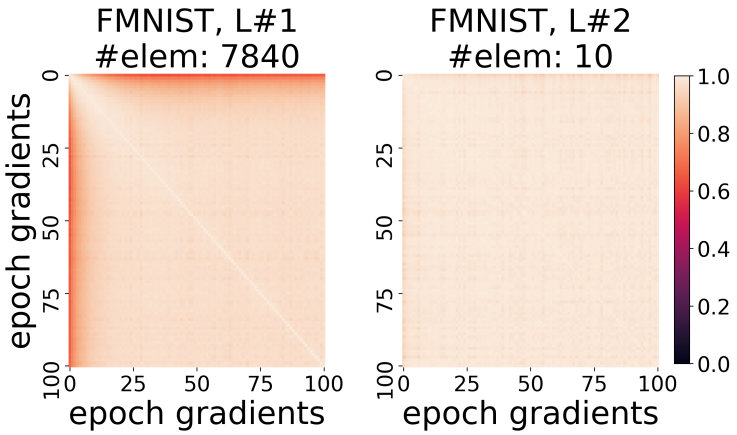

Figure 50: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **FCN** trained on **FMNIST** dataset.

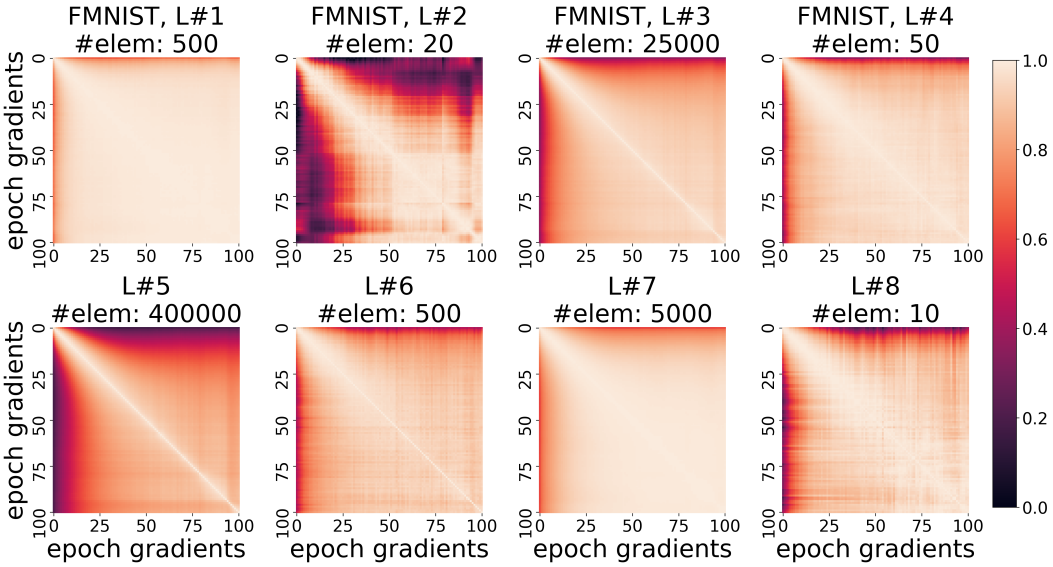

Figure 51: *PCA Components Overlap with Gradient*. Fig. 3 on **CNN** trained on **FMNIST** dataset.

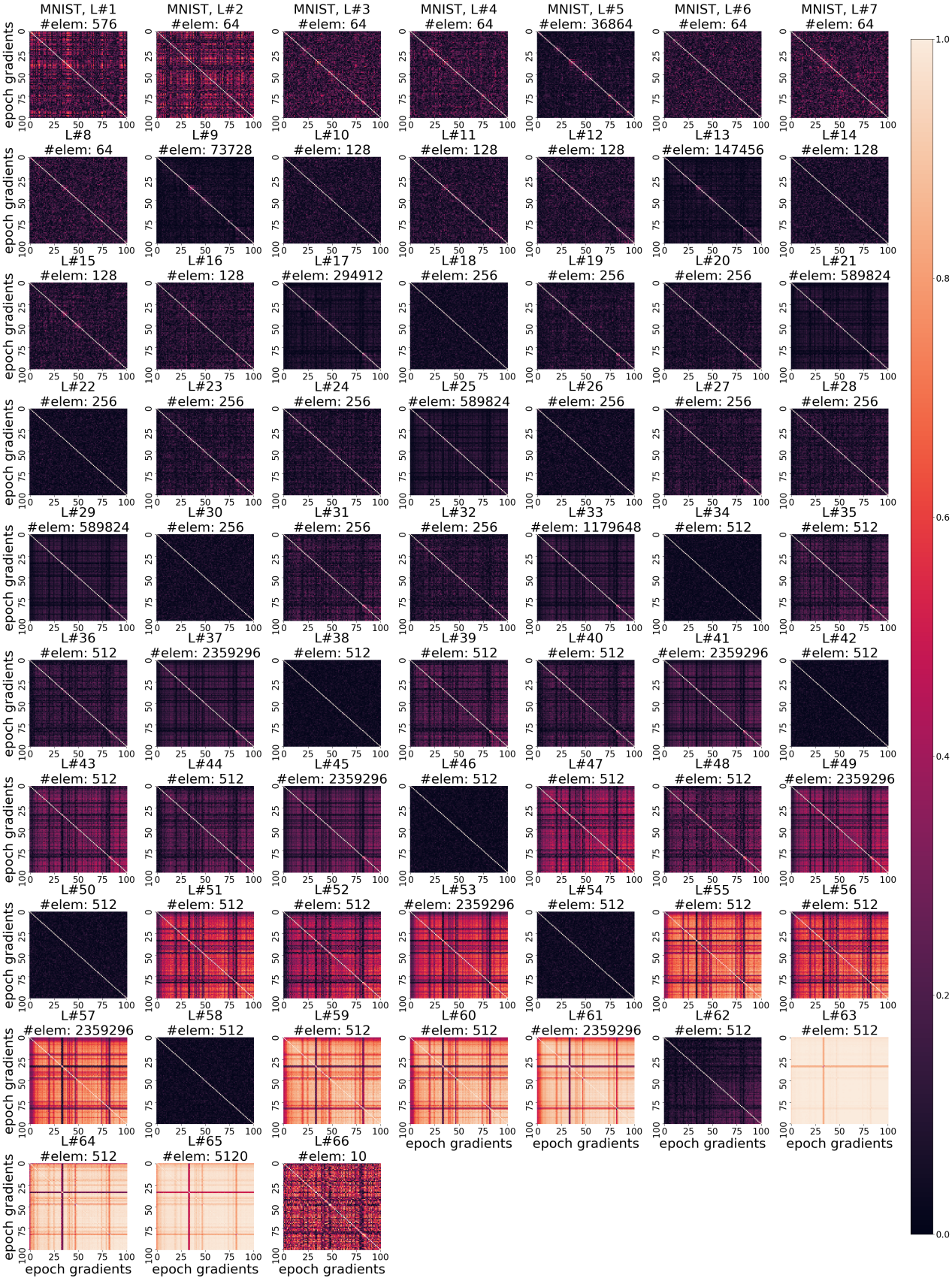

Figure 52: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **VGG19** trained on **MNIST** dataset.

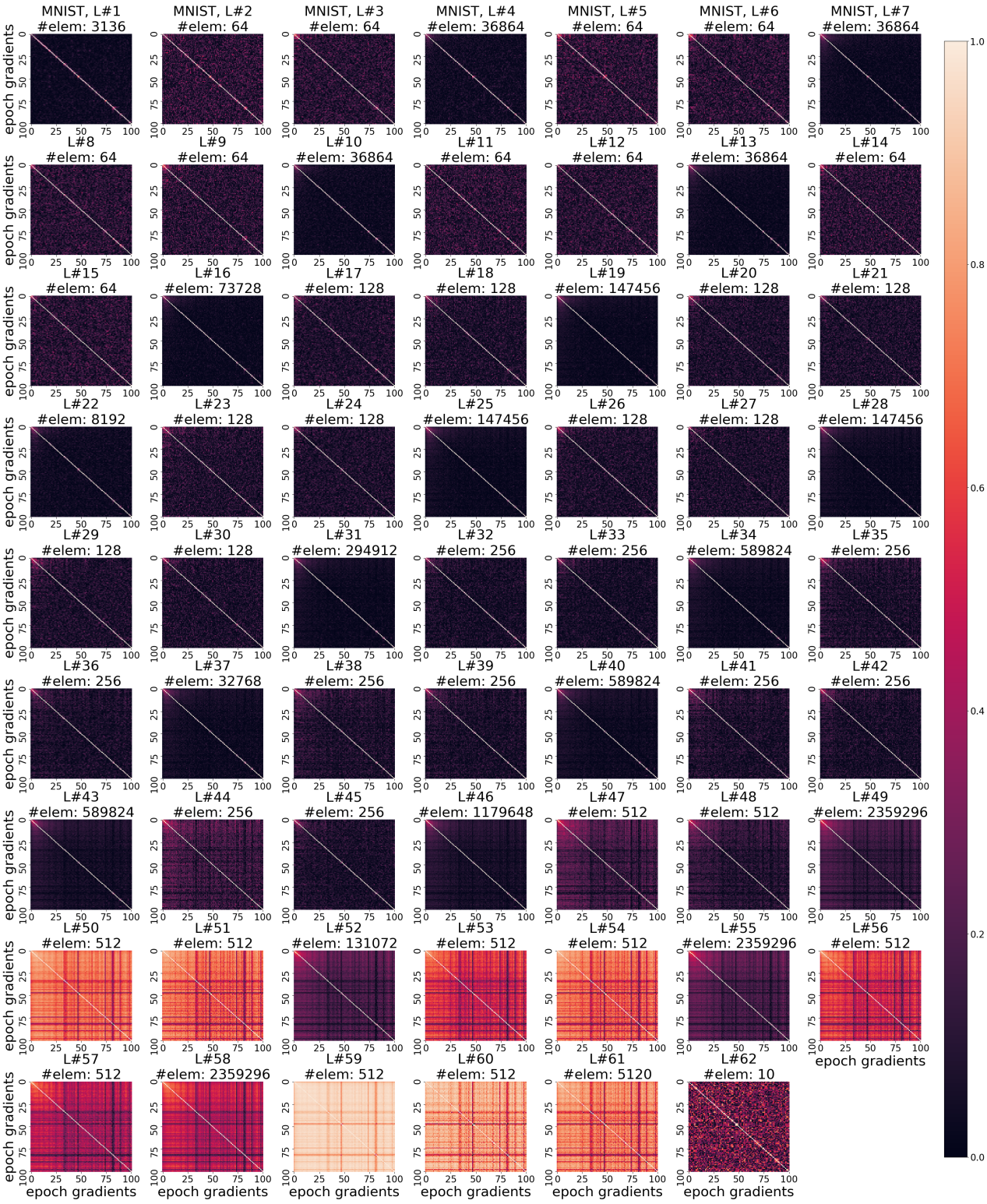

Figure 53: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **ResNet18** trained on **MNIST** dataset.

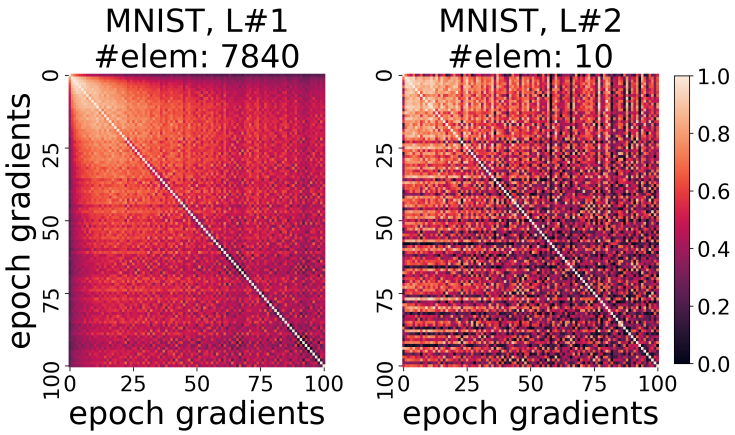

Figure 54: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **FCN** trained on **MNIST** dataset.

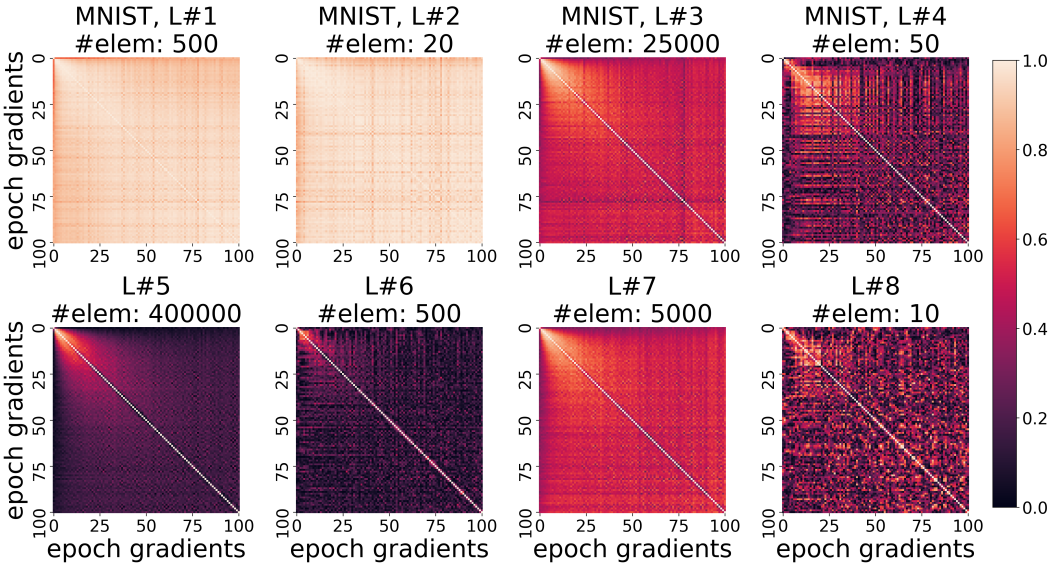

Figure 55: *PCA Components Overlap with Gradient*. Fig. 3 on **CNN** trained on **MNIST** dataset.

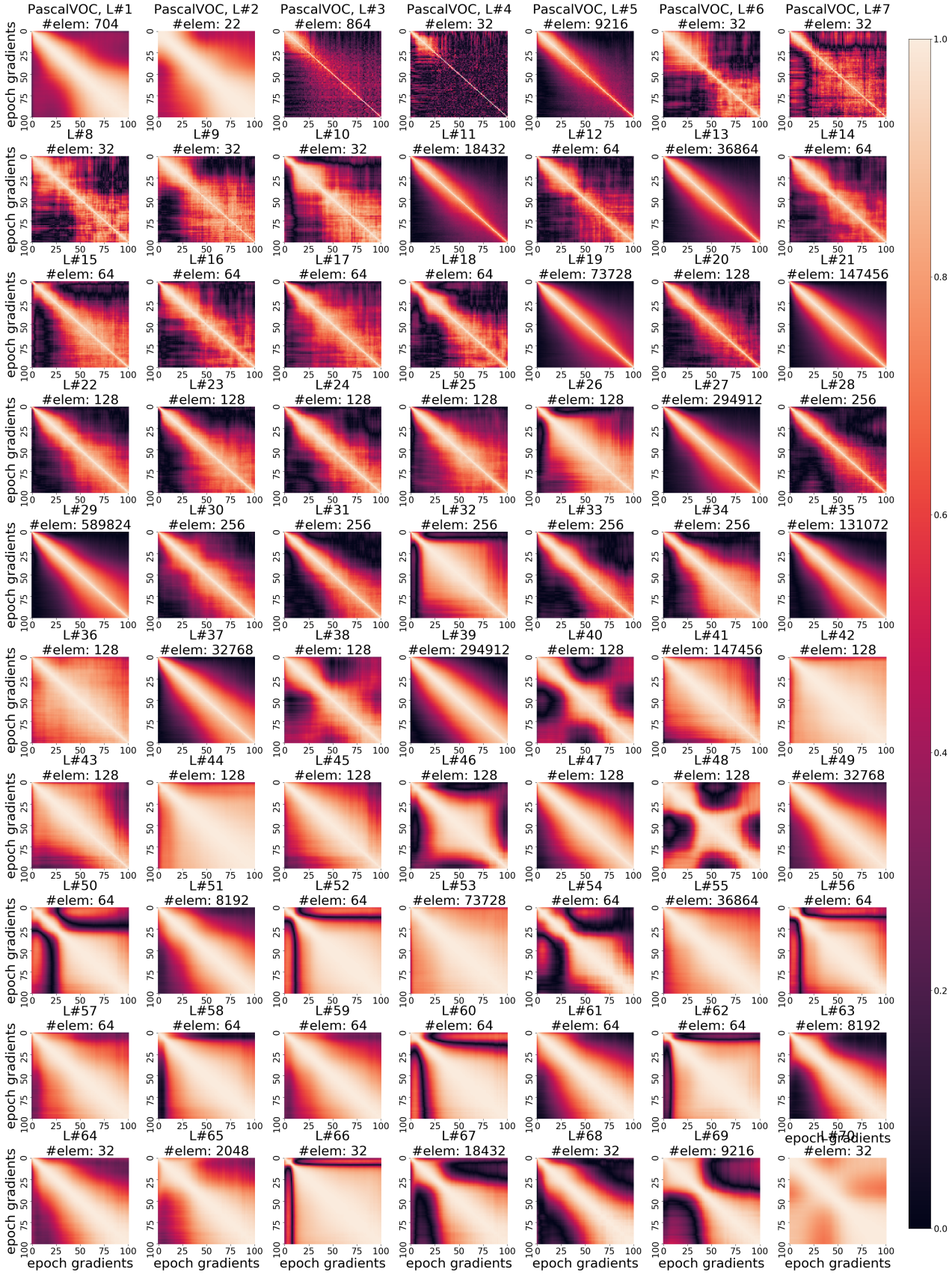

Figure 56: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **U-Net** trained on **PascalVOC** dataset.

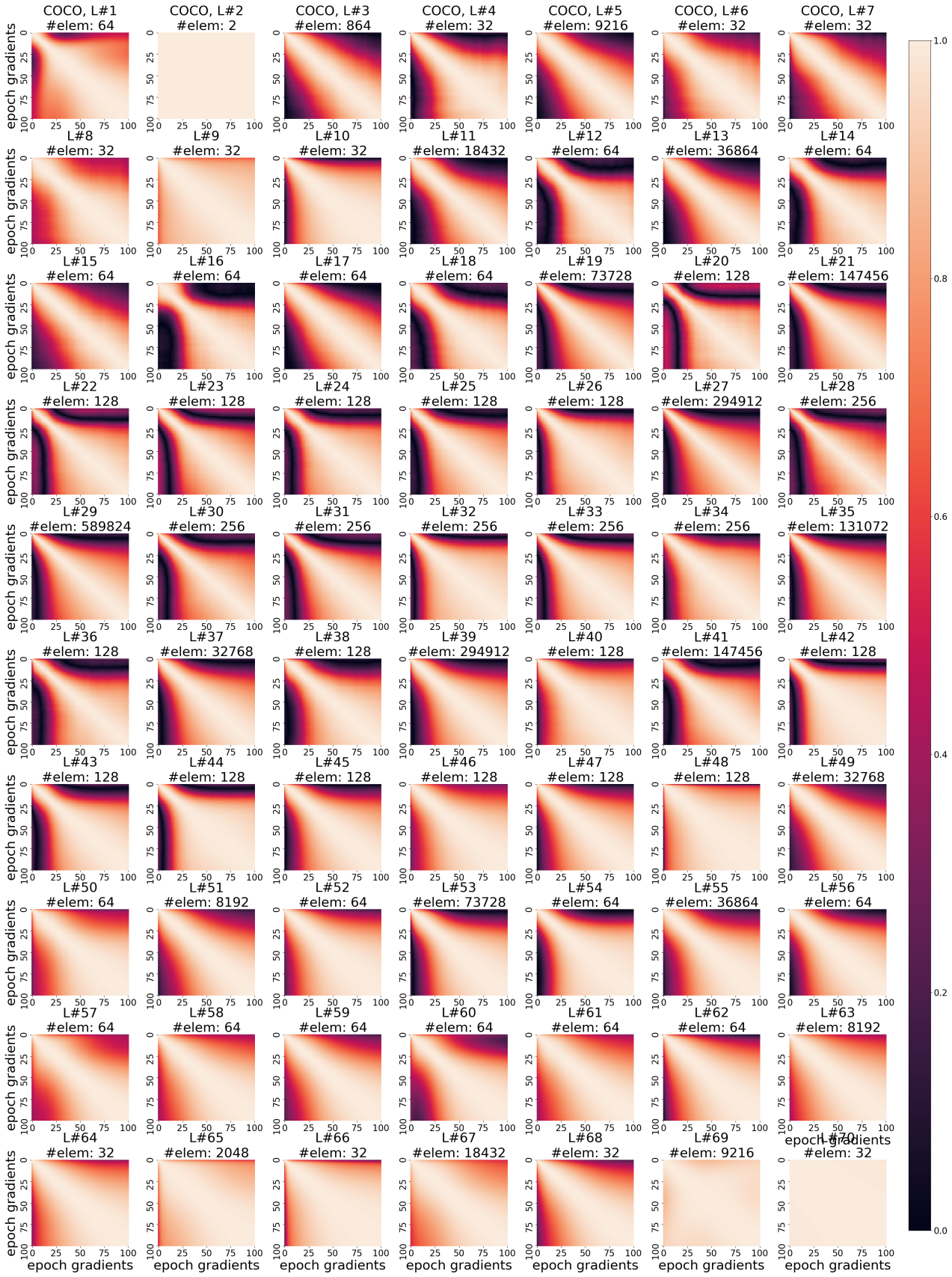

Figure 57: *PCA Components Overlap with Gradient*. Repeat of Fig. 3 on **U-Net** trained on **COCO** dataset.

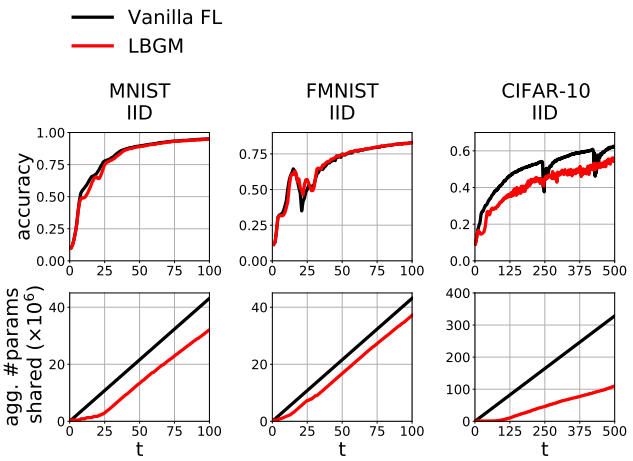

Figure 58: LBGM *as a Standalone Algorithm*. Experimental results in Fig. 5 repeated on datasets: **CIFAR-10**, **FMNIST**, and **MNIST** (iid data distribution) using classifier: **CNN**.

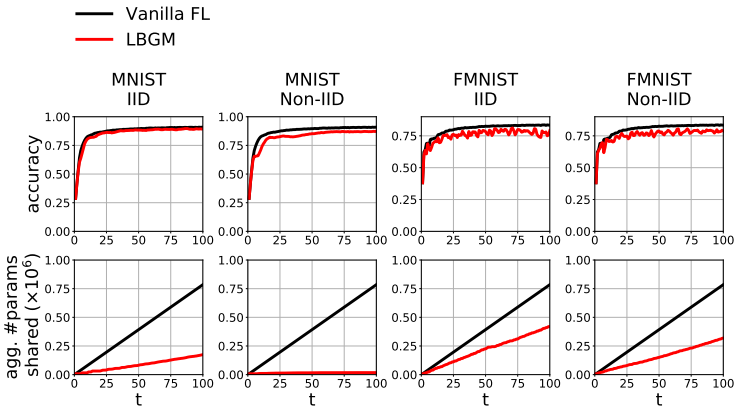

Figure 59: LBGM *as a Standalone Algorithm*. Experimental results in Fig. 5 repeated for datasets: **FMNIST** and **MNIST** (both iid and non-iid data distribution) using classifier: **FCN**.

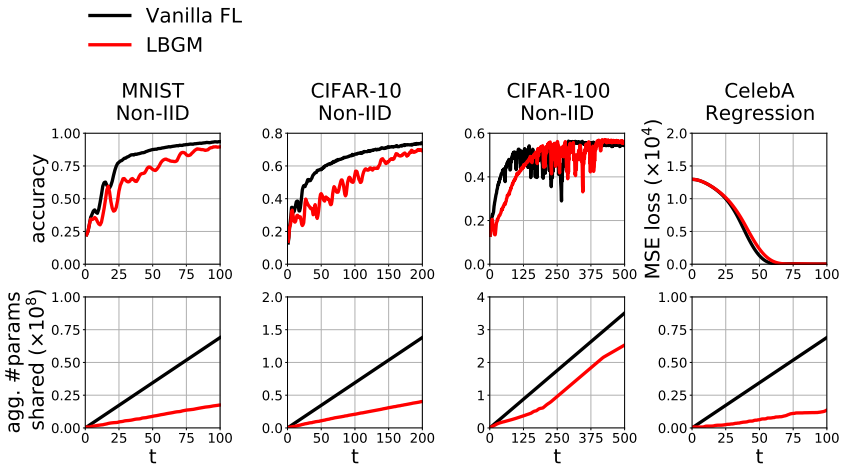

Figure 60: LBGM *as a Standalone Algorithm*. Experimental results in Fig. 5 repeated for datasets: **CIFAR-10**, **CIFAR-100**, **MNIST** (non-iid data distribution), and **CelebA** (face landmark regression task) using classifier: **Resnet18**.

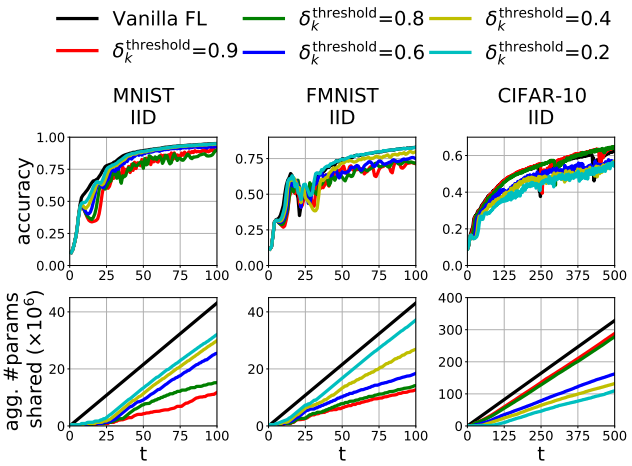

Figure 61: *Effect of* $\delta_k^{threshold}$ *on* LBGM. Experimental results in Fig. 6 repeated for datasets: **CIFAR-10**, **FMNIST**, and **MNIST** (iid data distribution).

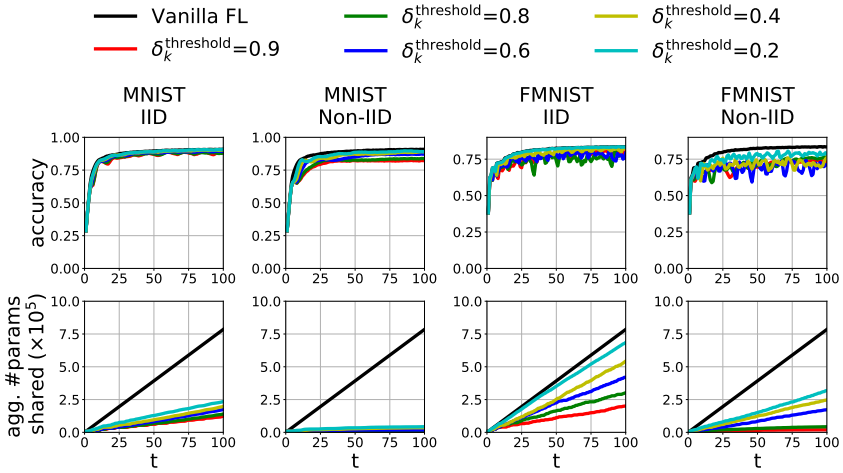

Figure 62: *Effect of* $\delta_k^{threshold}$ *on* LBGM. Experimental results in Fig. 6 repeated for datasets: **FMNIST** and **MNIST** (both iid and non-iid data distribution) using classifier: **FCN**.

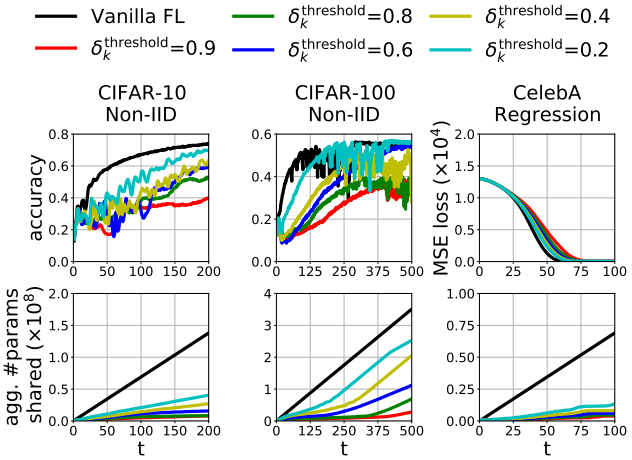

Figure 63: *Effect of* $\delta_k^{threshold}$ *on* LBGM. Experimental results in Fig. 6 repeated for datasets: **CIFAR-10**, **CIFAR-100** (non-iid data distribution), and **CelebA** (face landmark regression task) using classifier: **Resnet18**.

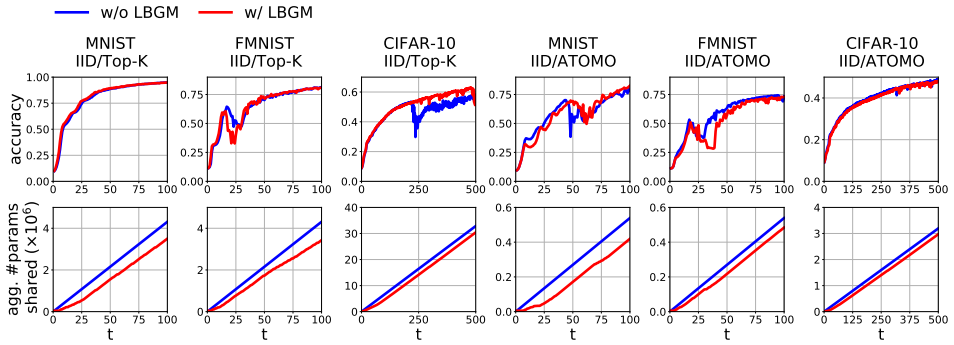

Figure 64: LBGM *as a Plug-and-Play Algorithm.*. Experimental results in Fig. 7 repeated for dataset: **CIFAR-10**, **FMNIST**, and **MNIST** (iid data distribution) using classifier: **CNN**.

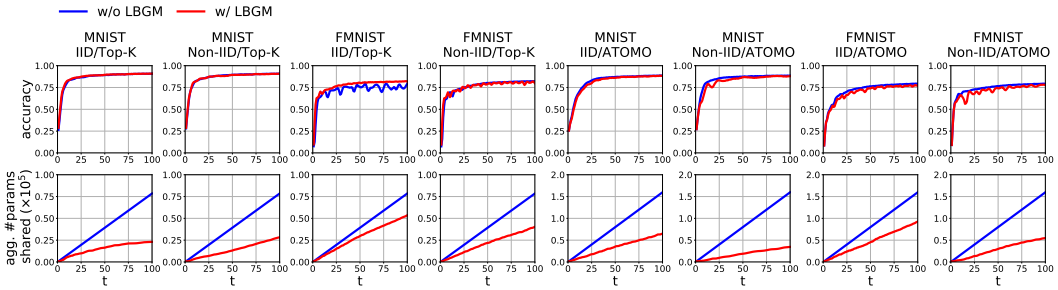

Figure 65: LBGM *as a Plug-and-Play Algorithm.*. Experimental results in Fig. 7 repeated for datasets: **FMNIST** and **MNIST** (both iid and non-iid data distribution) using classifier: **FCN**.

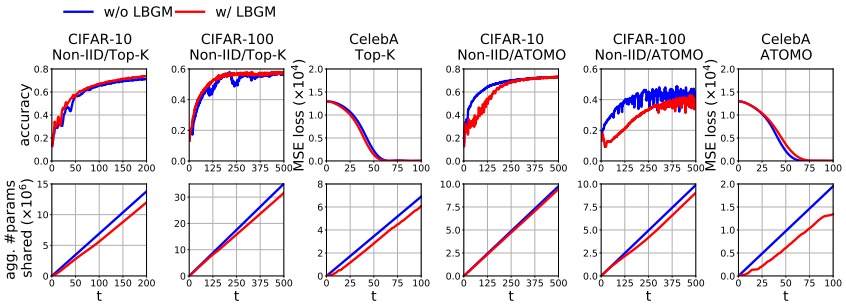

Figure 66: *Effect of* $\delta_k^{\text{threshold}}$ *on* LBGM. Experimental results in Fig. 7 repeated for datasets: **CIFAR-10**, **CIFAR-100** (non-iid data distribution), and **CelebA** (face landmark regression task) using classifier: **Resnet18**.

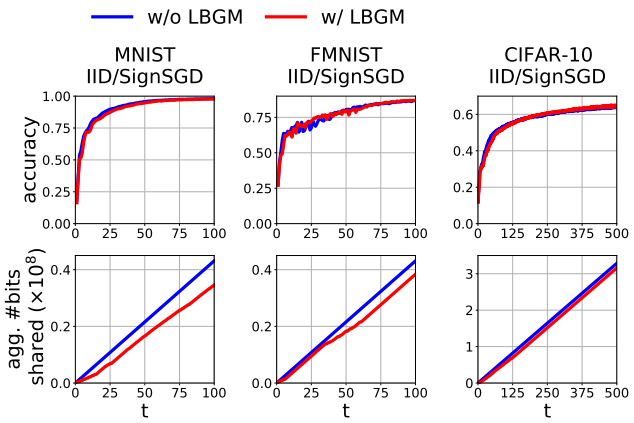

Figure 67: Experimental results in Fig. 8 repeated for dataset: **CIFAR-10**, **FMNIST**, **FMNIST** (iid data distribution) using classifier: **CNN**.

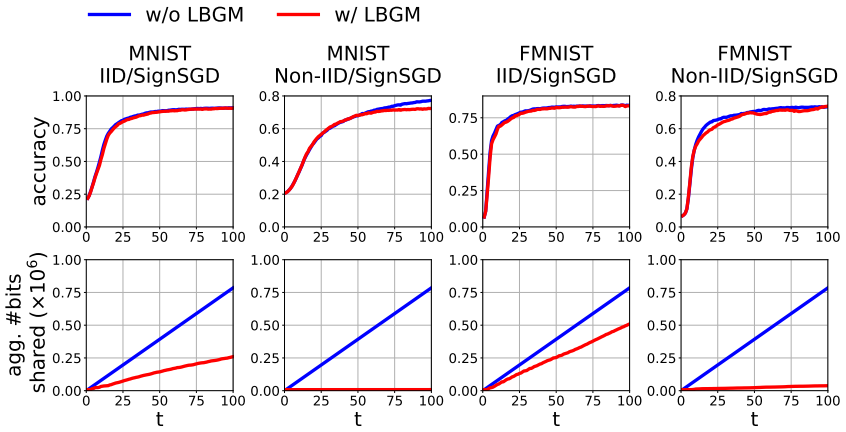

Figure 68: Experimental results in Fig. 8 repeated for dataset: **FMNIST** and **MNIST** (both iid and non-iid data distribution) using classifier: **FCN**.

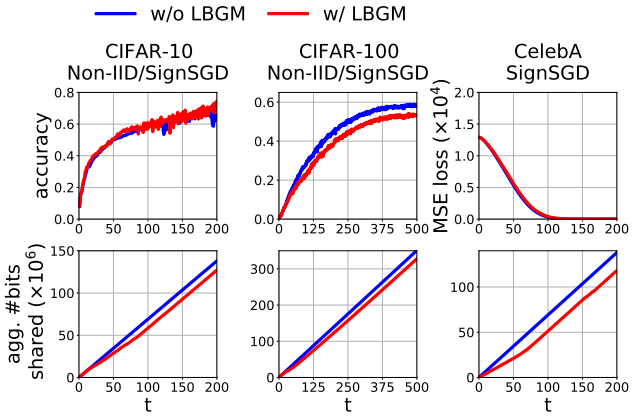

Figure 69: *Effect of* $\delta_k^{\text{threshold}}$ *on* LBGM. Experimental results in Fig. 8 repeated for datasets: **CIFAR-10**, **CIFAR-100** (non-iid data distribution), and **CelebA** (face landmark regression task) using classifier: **Resnet18**.