LQF: Linear Quadratic Fine-Tuning

Alessandro Achille[1] Aditya Golatkar[2,1] Avinash Ravichandran[1] Marzia Polito[1] Stefano Soatto[1]

[1]Amazon Web Services [2]UCLA

{aachille,ravinash,mpolito,soattos}@amazon.com aditya29@cs.ucla.edu

Abstract

Classifiers that are linear in their parameters, and trained by optimizing a convex loss function, have predictable behavior with respect to changes in the training data, initial conditions, and optimization. Such desirable properties are absent in deep neural networks (DNNs), typically trained by non-linear fine-tuning of a pre-trained model. Previous attempts to linearize DNNs have led to interesting theoretical insights, but have not impacted the practice due to the substantial performance gap compared to standard non-linear optimization. We present the first method for linearizing a pre-trained model that achieves comparable performance to non-linear fine-tuning on most of real-world image classification tasks tested, thus enjoying the interpretability of linear models without incurring punishing losses in performance. LQF consists of simple modifications to the architecture, loss function and optimization typically used for classification: Leaky-ReLU instead of ReLU, mean squared loss instead of cross-entropy, and pre-conditioning using Kronecker factorization. None of these changes in isolation is sufficient to approach the performance of non-linear fine-tuning. When used in combination, they allow us to reach comparable performance, and even superior in the low-data regime, while enjoying the simplicity, robustness and interpretability of linear-quadratic optimization.

1. Introduction

Deep neural networks (DNNs) are powerful but finicky. They can carve complex decision boundaries through high dimensional data such as images, but even small changes in the training set, regularization method, or choice of hyperparameters can lead to vastly different outcomes. This phenomenon, typical of highly non-linear optimization, is observed even when fine-tuning a pre-trained model, which is the most common *modus operandi* in practice: Starting from a DNN trained on some dataset, a few steps of stochastic gradient descent (SGD) are used to minimize a loss function computed on a another dataset. This is unlike models whose parameters are found via convex optimization, such

as support-vector machines: They have a global optimum, found from any initial condition, and small changes in the data, the regularization scheme, and hyperparameters yield small and interpretable changes in the final solution.

Lack of robustness to training conditions, and opaqueness of the resulting model, appear to be the price to pay for more performing and expressive classifiers such as a DNNs. This price is measured in time and cost of hyperparameter optimization (HPO). For example, simply changing the multiplier for weight decay requires retraining from scratch, as optimizing the new loss starting from the previous solution gives suboptimal results [11]. The complex relation between training data and final model makes it impossible to predict the effect of individual data, renders most generalization bounds vacuous, and makes it hard to impose even simple constraints, such as those arising from fairness criteria [20] or backward compatibility [44].

The desire to make their training more robust and interpretable has led some to *linearize* DNN models around an initial set of weights [28]. But while this has led to interesting theoretical insights, the analysis has failed to yield improvements in the practice. In particular, Figure 1 shows that linearized models perform large-scale image classification marginally better than simply training a linear classifier on a fixed pre-trained embedding. Non-linear fine-tuning with exhaustive HPO remains the performance paragon. The trade-off between performance and robustness, typical of many complex systems, is a manifestation of the classic bias-variance tradeoff: By reducing the sensitivity of the trained model to perturbations in the parameters or training data (variance, Figure 1, right), we incur a decrease in average performance (bias, left). While we do not expect linearized models to outperform non-linear fine-tuned ones – except in cases where the latter fails to optimize correctly (Section 4.5) – in this paper we explore how far we can push their accuracy, so we can enjoy the robustness and interpretability of linear models without punishing performance loss.

Our contribution, *Linear Quadratic Fine-Tuning* (LQF), is a method to linearize and train deep neural networks that achieves comparable performance to non-linear fine-tuning
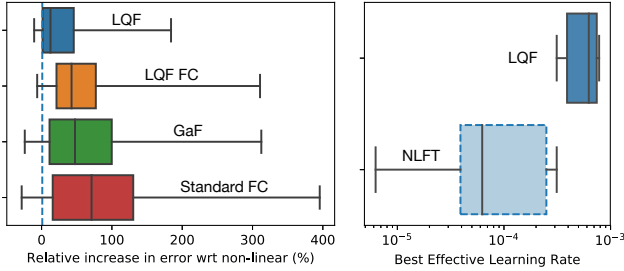
Figure 1: **Linear vs. nonlinear fine-tuning (NLFT). (Left)** Box-plot of the distribution of test errors achieved by different linearization methods on the datasets in Table 1, relative to the error achieved by NLFT (dashed line at origin). Whiskers show best/worst results, boxes extend from lower to upper quartiles (the results on half of the datasets are concentrated in the box), the central line represents the median increase in error. GaF [36] (green, 47% median error increase) is better than training a linear classifier on a fixed embedding (red, 71% increase), but slightly worse than LQF applied just to that linear classifier (LQF FC, orange, 42% increase). LQF is the closest linear method to the non-linear paragon (blue, 12% increase). **(Right)** We show the distribution of best effective learning rates as the task varies. While for NLFT we need to search in a wide range to find the optimal training parameters for a task (wide dashed box), for LQF the same learning rate works almost equally well for all tasks (narrow solid box).

(NLFT) on real-world image classification, while enjoying all the benefits of linear-quadratic optimization. LQF performs fine-tuning without optimizing hyper-parameters such as learning rate or batch size, enables predicting the effect of even individual training samples on the trained classifier, and easily allows incorporating linear constraints during training. LQF achieves performance comparable to NLFT, and better in the low-data regime which is the most relevant to many real applications.

The key enablers of LQF are simple and known in the literature, although not frequently used: (i) We replace the cross-entropy loss with the mean-squared error loss, making the optimization problem quadratic [13, 18, 3], (ii) we replace ReLU with Leaky-ReLU [33], and (iii) we perform pre-conditioning using Kronecker factorization (K-FAC) [35]. Individually, these changes bring limited improvements to standard training of non-linear models. However, we show that their combined use has a much larger impact on the performance of linearized models (Figure 2).

2. Related Work

Network linearization uses a standard first-order Taylor expansion of a pre-trained model (eq. 1). Work on the Neural Tangent Kernel (NTK) showed that linear models approxi-mate the dynamics of randomly initialized deep networks [19, 28], at least when the number of filters goes to infinity, although [36] argue the analysis is valid for finite networks in the case of fine-tuning when the weights w are close to those of the pre-trained model w_0. Using the linearized model amounts to training a linear model using the gradients of the original network as features, which is challenging given their dimension. However, [36] shows that this can be done efficiently with a modified forward pass using the Jacobian-Vector product algorithm. But while the training cost of the linear model is comparable to that of the original one, performance is not (Figure 1). [2] extends the approximation to higher orders and show increased fidelity, but loses the linearity of the model.

Loss function. We use the mean squared error (MSE) for fine-tuning instead of the more common cross-entropy (CE) loss, even when pre-training was performed with the latter. This may seem counter-intuitive, but there is growing evidence that the MSE can be competitive to train classifiers [13, 3, 18] and is just as well grounded theoretically [5]. There is also evidence that the standard cross-entropy loss learns representations that are more transferable [26], and hence it is better suited for pre-training, even if it is not necessarily the best performing loss in terms of test accuracy. Our use of CE for pretraining and MSE for fine-tuning fits in this paradigm.

Pre-conditioning is important to converge on badly conditioned convex problem. We use Kronecker-factorized (K-FAC) approximation of the Fisher [35, 8, 15] to efficiently approximate the curvature of the loss function. Since curvature is constant in LQF, we only need to compute the preconditioning matrix once, sensibly reducing its cost.

Hyper-parameters can significantly affect performance in NLFT, which requires careful optimization (HPO) of learning rate, momentum and batch size based on the source and target task [29]. A linear model does not require extensive HPO (Figure 1). Being strongly convex, LQF has a unique global minimum to which SGD will provably converge, assuming a proper learning rate schedule is used [38].

Interpretability. We study how a given training sample affects the learned model and its predictions. This analysis is similar to that of *influence functions* [25, 4], but in our case can be done without approximation since our loss is quadratic. Unlike influence functions, we can also efficiently compute the change of activations on a validation set, which we use to compute informativeness (Section 3.4).

Applications. Having a linear model that performs on-par with NLFT makes it possible to tackle open issues in large-scale DNNs, such as (i) backward-compatibile training without model distillation [44], (ii) leave-one-out test error estimation without a test set [9] (iii) measuring the influence

of individual samples on the trained model [25, 17] to enable active learning and enforce privacy guarantees [7], (iv) integrating the model in a continual learning framework (Section 4.5).

3. Linear-Quadratic Fine-tuning

Linearized model. LQF addresses the lack of robustness and interpretability of non-linear deep models by replacing the network itself with a first-order linear approximation. Let $f_w(x)$ denote the output of a deep network with weighs w on an input image x, let w_0 denotes an initial set of weights, for example obtained after pre-training on ImageNet. We consider the *linearization* $f_w^{\text{lin}}(x)$ of the network $f_w(x)$ given by the first-order Taylor expansion of $f_w(x)$ around w_0:

$$f_w^{\text{lin}}(x) = f_{w_0}(x) + \nabla_w f_{w_0}(x) \cdot (w - w_0). \quad (1)$$

Note that $f_w^{\text{lin}}(x)$ is linear with respect to the weights w; however, due to the non-linear activation functions, it is still a highly non-linear function of the input x. If the approximation is accurate, ideally we want the linearized f_w^{lin} and the original f_w to reach similar accuracy when trained:

$$\text{accuracy}(f_{\hat{w}}^{\text{lin}}, \mathcal{D}_{\text{test}}) \approx \text{accuracy}(f_w, \mathcal{D}_{\text{test}}) \quad (2)$$

where $\hat{w}$ and w are obtained by minimizing, respectively, the loss of the linear and non-linear model on a training set $\mathcal{D}$. The obvious advantage of the linear model is that, for strongly convex loss functions, the global optimum $\bar{w}$ is unique and, for a quadratic loss functions, can even be written in closed form. So, if eq. (2) was satisfied, we would stand to gain in terms of speed of fine-tuning, performance (convergence is guaranteed to the global optimum), and interpretability (the effect of a training sample on the test prediction can be computed exactly).

Training a linearized model requires computing the product of the Jacobian $\nabla_w f_{w_0}(x)$ with the weights w, which would require a separate backward pass for each sample. However, [36] shows that using Gradients as Features (GaF) the Jacobian-Vector product of eq. (1) can be computed with a modified forward pass, at a cost comparable to that of the running the original network.

Accuracy of linearized models. In Figure 1 and Table 1 we show that the accuracy of linearized models suffers compared to the paragon of NLFT. In most datasets, even the simple linear baseline obtained by fine-tuning just the linear classifier (fully-connected, or FC, layer) is only slightly worse than GaF, violating (2). We now analyze the causes of these differences between the dynamics of fine-tuning a non-linear network and its linearized version, and propose a set of changes that lead to LQF.
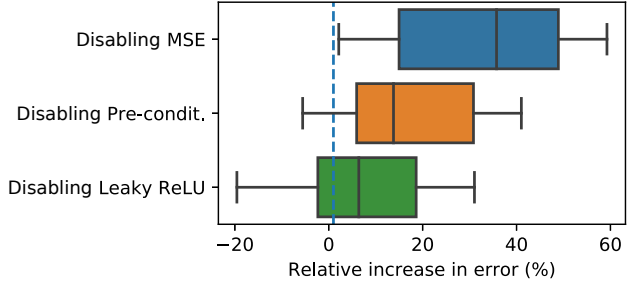

Figure 2: **Ablation study.** Box plot showing relative error increase on the datasets in Table 1 when removing constitutive components of LQF. Removing MSE in favor of the standard CE loss yields the largest increase in error (blue). Removing K-FAC pre-conditioning, leaving standard SGD (orange), has also a sizeable effect. Finally, on most datasets, Leaky-ReLU performs better than standard ReLU (green).

3.1. Loss function

We use a regularized mean-squared error (MSE) loss, even if the original model is pre-trained by minimizing the regularized cross-entropy loss:

$$L_{\text{MSE}}(f; \mathcal{D}) = \sum_{(x_i, y_i) \in \mathcal{D}} \|\alpha y_i - f(x_i)\|^2 + \frac{\lambda}{2} \|w - w_0\|^2, \quad (3)$$

where y_i is the one-hot vector encoding the class label and α is a scaling factor for the targets (we set $\alpha = 15$ as in [18]).[1] There are several reasons to make this change. First, while standard DNNs have several normalization layers (such as batch normalization) that keep the output bounded, the linear model $f_w^{\text{lin}}(x)$ in eq. (1) directly outputs the dot product $\nabla_w f_{w_0}(x) \cdot w$, which involves tens of millions of weights. This product can easily grow large and saturate the softmax of the cross-entropy loss, making the training process unstable. When that happens, we observe that only the last layer features are effectively trained. On the other hand, the MSE loss does not saturate and forces the output to remain close to the target one-hot vectors, and hence bounded. We also note that using MSE gives better results *even* for standard non-linear training (see Appendix), especially when training on small datasets.

A secondary effect of using the MSE loss in conjunction with a linear model is that we can write the unique global optimum in closed form:

$$w^* = (J^T J + \lambda I)^{-1} J(Y - f_0(X)) \quad (4)$$

where $J \equiv \nabla_w f_{w_0}(X)^T$ is the matrix of the Jacobians of all the samples in the training set and Y is the matrix of all target

---

[1]Note that we penalize the distance of the weights w from the pretraining w_0. This is more natural than the weight decay penalty $\frac{\lambda}{2}\|w\|^2$ in our setting, since we are optimizing for the perturbation $w - w_0$. It may also lead to better transfer results on some tasks [30].

vectors. While w^* cannot be computed directly using this formula due to large size of the matrices, in Section 3.4 we show that eq. (4) allows us to easily compute the influence of a given training sample on the test predictions, enhancing interpretability of the training process.

3.2. Pre-conditioning with K-FAC

The MSE loss in eq. (3) is strictly convex, so SGD is guaranteed to converge to the unique global minimum, provided an appropriate learning rate decaying schedule is used [38]. However, different directions in the loss landscape may have different curvature, slowing convergence: the same learning rate may be too fast high-curvature directions and too small for flat ones. Convergence speed is governed by the condition number of the Hessian matrix (ratio between the largest and smallest eigenvalues, i.e., maximum and minimum curvature), which can be modified by multiplying the update with a *pre-conditioning* matrix A_t

$$w_{t+1} = w_t - \eta A_t g_t, \tag{5}$$

where g_t is the batch gradient computed at step t and η is the learning rate. The matrix A_t allows training at different speed in each direction. In a quadratic problem with constant preconditioning $A_t = A$, the expected distance from the optimum w^* at time t using adaptive SGD is given by:

$$\mathbb{E}[w_t - w^*] = (I - \eta A H)^t (w_0 - w^*), \tag{6}$$

where H is the (constant) Hessian of the loss function. In particular, the fastest convergence is obtained when $A = H^{-1}$ is the inverse of the hessian. For this reason, most adaptive methods try to approximate H^{-1}.

Adaptive methods are commonplace in convex optimization and some [23] are widely used in deep learning. However, for large-scale visual classification, standard SGD with momentum performs comparably or better than adaptive methods. We hypothesize that this may be due to the changing curvature during training, which causes the condition number to decrease, making the problem well-conditioned even without adaptation (see Figure 3). However, this cannot happen when training a linearized network, since the curvature of eq. (3) is constant and fully determined in pre-training. For this reason, we add a pre-conditioner for eq. (3) in LQF.

In our case, since the problem is quadratic, the Hessian coincides with the Fisher Information Matrix (FIM). We thus use the Kronecker-Factorized (K-FAC) approximation of the FIM as pre-conditioner [35]. Unlike diagonal approximations, K-FAC allows us to estimate off-diagonal terms while remaining tractable.

K-FAC is efficient for LQF. On standard non-linear networks, K-FAC may make the weights converge too quickly to suboptimal sharp minima that do not generalize well and,
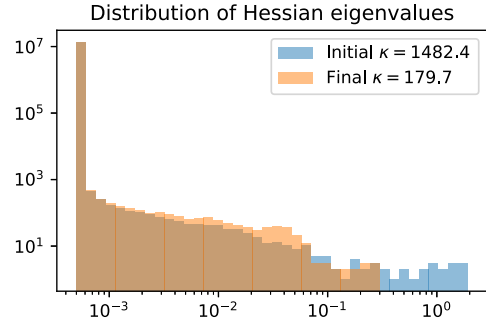

Figure 3: **Unlike SGD, LQF requires pre-conditioning.** Histogram of the eigenvalues of the Hessian loss function at initialization (blue) and at the end of optimization of a non-linear network (orange). Note that at initialization a few directions have very large eigenvalues while many directions have much smaller eigenvalues, resulting in a high condition number $\kappa = 1482.4$. However, SGD naturally moves to areas of the loss landscape that are better conditioned, with $\kappa = 179.7$ at the end of training (note that the largest eigenvalues decrease), thus making convergence easier. This "automatic conditioning" makes pre-conditioning un-necessary in non-linear networks. On the other hand, since in LQF the Hessian is fixed throughout the training, we need to use K-FAC pre-conditioning to facilitate convergence.

since the curvature changes over time, the approximation of the Hessian needs to be updated. However, in LQF there is a unique minimizer and the curvature is constant. This makes the use of K-FAC especially efficient in our setting. Compared to other adaptive methods like Adam [23], in our setting K-FAC provides an accurate estimation of the Hessian inverse which can be used to interpret the effect of a training sample (Section 3.4) and to easy select an appropriate learning rate.

Learning rate selection. From eq. (6) it follows that, to guarantee convergence, the norm of the eigenvalues of $I - \eta A H$ should be smaller than one. In particular, if A is a good approximation of H, the eigenvalues of AH are centered around 1 and $\eta = 1$ is optimal. In our case A will not be a perfect approximation of H^{-1}, however since we expect $\lambda_{\max}(A^{-1}H) \approx 1$ and we need $\eta < 2/\lambda_{\max}(A^{-1}H)$ to guarantee convergence, we choose a conservative $\eta = 0.1$, which we observe gives consistent performance on all datasets without further tuning (Figure 1, right).

3.3. Leaky-ReLU activations

Rectified linear units (ReLUs) [37, 46] leave the positive component of the input unchanged and replace the negative component with zero, which blocks information from certain weights. This is not harmful during normal training, since

batch normalization helps avoiding dead filters. However, in linearized models only the gradient of the filter at initialization matters in computing the Jacobian-Vector Product. If this is zero, it will stunt training for that filter. To prevent this, we simply replace ReLU with Leaky-ReLu [33], which does not annihilate the negative component but simply replaces it with a value proportional to the input. Another advantage of Leaky-ReLU is that its first order approximation near zero is more accurate than that of ReLU, making the first-order approximation in eq. (1) more accurate.

Performance of LQF. After accounting for these changes, we repeat the previous experiments on different datasets and report the overall results in Figure 1 and detailed in Table 1. LQF outperforms other linearization methods on all the datasets, and achieves performance comparable to non-linear fine-tuning on most. In the low data regime, the strong inductive bias of the linear model acts as a regularizer and the linearized model outperforms the non-linear fine-tuned one (Figure 4). We discuss the results in detail in Section 4.

3.4. Interpretability

LQF gives us an efficient way of estimating the influence, or "effect" of a given training sample on the final weights and on the test predictions of the DNN, as well as measuring the informativeness to the training process. This can be used for dataset summarization (Sect. 4.6) and to interpret model predictions: Had a particular sample (cause) not been used for training, how would the final model (effect) differ? Using eq. (4) we have the following expression for the weights w_{-i} we would have obtained if training without the sample x_i:

$$w^*_{-i} = w^* + (F_{-i} + \lambda I)^{-1} g_i. \quad (7)$$

where $g_i \equiv \nabla_{w^*} f_w(x)$ is the Jacobian of the i-th sample, $e_i \equiv \frac{1}{N}(y - w^* g_i)$ is the weighted prediction error on the sample x_i, N is the number of training samples, and F_{-i} is the hessian (or, equivalently, the FIM) computed without x_i.

Using eq. (7) we can estimate how much the prediction $f_w(x_{\text{test}})$ on a test sample x_{test} would change had we not seen the training sample x_i. That is, we can measure which training sample has contributed more to a correct or incorrect decision of the network. Using eq. (7), in the Appendix we show that

$$f_{w^*}(x_{\text{test}}) - f_{w^*_{-i}}(x_{\text{test}}) \approx$$
$$\left(1 - \frac{\alpha_i}{N - 1 + \alpha_i}\right) e_i g_{\text{test}}^T (F + \lambda I)^{-1} g_i, \quad (8)$$

where $\alpha_i \simeq g_i(F + \lambda I)^{-1} g_i^T$. Since K-FAC already provides an approximation of $(F + \lambda I)^{-1}$, we can estimate the expression efficiently.

More generally, one can define the unique functional information [17] of a training sample x_i as the expected

influence that the sample has on the network output on a validation set:

$$\text{F-SI}((x_i, y_i)) = \mathbb{E}_{x \sim \mathcal{D}_{\text{val}}} \left[\| f_w(x) - f_{w_{-i}}(x) \|^2 \right] \quad (9)$$

This measure – which can be used for dataset summarization and to interpret what the network learn from a training samples (Section 4.6) – is also easily estimated using eq. (8).

4. Results

We test LQF on several standard vision classification tasks used to benchmark fine-tuning learning [30, 29] and additional tasks to test cross-domain transfer (see Appendix for details on the datasets). We find that LQF outperforms other linearization methods on almost all tasks. LQF also outperforms standard non-linear training (NLFT) in tasks such as few-shot learning and on-line learning, where scarce data causes training instabilities. We also illustrate the benefits of interpretability.

Architecture and training. As the base model for linearization we use a ResNet-50 with Leaky-ReLU pre-trained on ImageNet. We implement the forward pass of eq. (1) using [36], and minimize the MSE loss, with K-FAC preconditioning (using the implementation of [8]). For NLFT, we also use a pre-trained ResNet-50 with ReLU activations, and minimize the cross-entropy loss. All experiments are conducted with SGD with momentum 0.9, learning rates $\eta \in \{0.01, 0.001\}$, weight decay $\in \{10^{-4}, 10^{-5}\}$ and report the best results. We discuss further training details in the Appendix.

4.1. Comparison

Ideally, LQF should achieve close performance to NLFT on all tasks. We also compare with two other linearization methods: Standard FC, which uses the pre-trained network as a feature and only retrains the last FC layer, and Gradients as Features (GaF) [36] which linearizes the network without any of the changes described in Section 3. We also introduce a further baseline, LQF FC, which uses the changes in Section 3, but only to train the final FC layer.

We train each model on several "coarse-grained" datasets, covering different tasks and domains and report the results in Table 1. In Figure 1 we show the distribution of error of each methods relative to the NLFT paragon. We see that GaF improves only marginally over the FC baseline, and is usually far from the non-linear performance. On the other hand, LQF performs comparably with NLFT on all the coarse-grained datasets. Surprisingly, training only the final classifier using LQF (LQF FC) improves over Standard FC, suggesting that the changes discussed in Section 3 are helpful in broader scenarios.

	NLFT	LQF*	LQF FC*	GaF	FC
Caltech-256 [14]	14.1	**14.5**	17.4	15.4	15.7
Chest X-Ray [21]	4.6	**7.1**	8.3	9.5	11.2
Malaria Cells [43]	3.2	**4.1**	5.5	4.8	6.4
MIT-67 [42]	20.5	**20.7**	24.7	23.1	24.3
Oxford Pets [41]	6.7	**6.9**	7.7	7.4	7.7
Fine-grained datasets (sorted by ImageNet distance)					
Stanf. Dogs [22]	13.8	12.4	12.9	10.5	**9.9**
Ox. Flowers [40]	7.1	**7.1**	9.7	13.3	13.6
CUB-200 [47]	19.6	**24.0**	29.1	28.4	29.2
Aircrafts [34]	14.5	**34.5**	43.8	48.4	54.2
Stanf. Cars [27]	9.6	**27.1**	39.2	39.4	47.3

Table 1: **Test error of linear and non-linear fine-tuning on coarse-and fine-grained classification**. On almost all datasets LQF outperforms all other linear methods. Moreover, on most coarse datasets LQF is within 0.5% absolute error from standard non-linear fine-tuning. Linear methods performs worse than NLFT on fine-grained machine classification tasks that have a large domain gap with respect to ImageNet pretraining – e.g., *Stanford Cars* and *FGVC Aircrafts*. However, even in this case LQF reduces the error by up to 20% with respect to competing methods.

4.2. Fine-grained classification and Bilinear Pooling

Since LQF is based on a first-order Taylor expansion, it has an implicit bias toward remaining close to the pre-training (ImageNet in our case). [29] notes that such a bias may improves accuracy on tasks that are easier or nearby (e.g., Stanford Dogs, Caltech-256), but it is detrimental on fine-grained tasks with a large domain gap from ImageNet (e.g., CUB-200, Aircrafts, Cars). We expect the same to hold LQF. Indeed, in Table 1 we show that while performance is close or better on similar tasks, it degrades for farther tasks (sorted as in [29]). However, even in this case, we observe that LQF significantly outperforms other linearization techniques.

B-LQF. These results suggest that finding a pre-training closer to the target task may help improve the linearizaion performance [1, 39, 6], which is indeed an intereting direction of research. On the other hand, [32] suggests that even for generic ImageNet pretraining, the covariance of the last-layer features, rather than their mean, is more suited as a feature for fine-grained visual classification. Following this intuition, we replace the last layer global spatial mean pooling with square root bilinear pooling: if $z_{i,j}$ are the features of the last-layer feature map, instead of feeding their mean $\mu = \mathbb{E}_{i,j}[z_{i,j}]$ to the fully connected layer, we feed the square root $\sqrt{\Sigma}$ of their covariance matrix Σ [32, 31, 8]. In Table 2 we see that – while bilinear pooling does not improve the performace of NLFT too much – combining

	NLFT	MPN-COV	B-LQF*	LQF*	GaF
CUB-200	19.6	17.0	**17.1**	24.0	29.2
Aircrafts	14.5	15.4	**29.2**	34.5	54.2
Cars	9.6	9.1	**16.3**	27.1	47.3

Table 2: **Fine-grained classification and bilinear pooling.** Normalized bilinear pooling (MPN-COV) [31, 8] does not significantly improve ordinary classification results (NLFT) in our setting. However, LQF with bilinear pooling (B-LQF) significantly improves test accuracy compared to average pooling (LQF).

bilinear-pooling with LQF (B-LQF) significantly boosts its performance on fine-grained tasks.

4.3. Ablation study

We measure the effect of each component of LQF on its final accuracy. In Figure 2 we show the relative increase in error when we ablate the changes discussed in Section 3. Using the MSE loss instead of cross-entropy yields the largest improvement. Using K-FAC also carries significant weight in the final solution. This is interesting since, in principle, SGD without pre-conditioning should recover the same minimum. This reinforces the point that the optimization problem is badly conditioned (Figure 3), and proper pre-conditioning is necessary. Using Leaky-ReLU activations instead of ReLUs also improve the final results on average, even if by a lesser margin than the other two changes. However, in additional ablation studies where we ablate pairs of components, we observe that using Leaky ReLU makes a larger difference if K-FAC is not present, possibly because Leaky ReLU simplifies the optimization of the linear model and is more useful when K-FAC is disabled.

4.4. Low-shot data regime

LQF, like any linearization, cannot be expected to outperform NLFT in general. However, there may be scenarios where it does systematically. In the low-data regime, non-linear networks can easily memorize samples using spurious correlation without having a chance to converge to fit relevant features. On the other hand, LQF should have a strong inductive bias arising from the existence of a unique global minimum. To test this, in Figure 4 we plot the test error of LQF relative to the NLFT paragon. In the k-shot scenario (i.e., training using only k samples per class) LQF outperforms NLFT systematically. This suggests that LQF may be a better base model than NLFT in few-shot learning, also because it facilitates the incorporation of any convex regularizer.
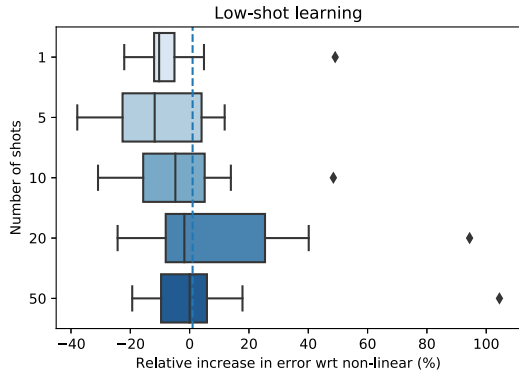
Figure 4: **Efficiency of LQF at different data regimes.** Comparison of LQF and standard fine-tuning for different number of training samples per category ("shots") in the same datasets as Table 1. On average, LQF performs better than standard fine-tuning when the datasets are relatively small. The single outlier (diamond) is FGVC Aircraft, where NLFT tends to perform better (see Section 4.2).

4.5. On-line learning

In many applications, training data is not all available at the outset. Instead, it trickles in continuously and triggers re-training upon reaching a sufficiently large volume. This protocol engenders tradeoffs between cost and obsolescence. Fine-tuning with relatively small batches is a typical compromise, but can lead to suboptimal solutions when the network memorizes the new samples or catastrophically forgets old ones. LQF is not subject to this tradeoff as it has a unique minimum that is adjusted incrementally as new data is received.

In Figure 5, we compare incremental LQF incremental re-training with incremental NLFT training and with the paragon of re-training from scratch every time we obtain more data on the MIT-67 dataset. Let $\mathcal{D}_t$ be the dataset containing all the samples seen up to step t; we train on $\mathcal{D}_t$ starting from the weights w_{t-1} obtained at the previous step and stop training when the train error on $\mathcal{D}_t$ is below 0.5% for 5 consecutive epochs. In Figure 5, we split all the samples of MIT-67 in 30 incremental steps (each step containing 178 samples sampled uniformly from all classes) and show the test accuracy of the model at different steps. As expected, incremental NLFT performs substantially worse than the paragon, as it gets stuck in suboptimal local minima. On the other hand, LQF is always close to the paragon even if trained incrementally while being much cheaper to train since we do not need to retrain from scratch.

4.6. Informativeness of samples

LQF is interpretable in that it allows measuring the effect of individual data on the trained model, which can be used

to summarize a dataset. We use eq. (9) to measure how informative a training sample is for the predictions of the final model. In Figure 6, we display the most and least informative samples: Qualitatively, we observe that easy samples and near-duplicates are not informative. On the other hand, samples of classes that are easy to confuse are considered very informative by the model, since they affect the decision boundary. To test this, in Figure 6 (right) we plot the test error of the model when the most/least informative training samples are removed. As we expect, removing informative samples degrades the performance more than removing uninformative samples.

4.7. Robustness to optimization hyper-parameters

Non-linear fine-tuning is highly sensitive to the choice of hyperparameters (Figure 1, right). This has a direct impact on cost, as training requires broad hyper-parameter search. It would be desirable if one could fix hyperparameters once for fine-tuning. In Section 3.2 we argued that, due to the use of MSE loss and K-FAC pre-conditioning, the optimal learning rate is independent of the task for LQF. When using SGD with momentum, the situation is slightly more complex, as we also need to select batch size b and the momentum m. As already noted by [29], when fine-tuning, different runs typically have the same test accuracy as long as the effective learning rate ELR $= \frac{\eta}{(1-m)b}$ is the same. We claim that, for LQF, there is a single ELR which is optimal for most tasks. To test this, we search the best learning rate $\eta \in \{0.05, 0.01, 0.005, 0.001, 0.0001\}$ and batch size $b \in \{16, 32, 64\}$, and compute the ELR of the best combination for each task. In Figure 1 (right) we report the distribution of
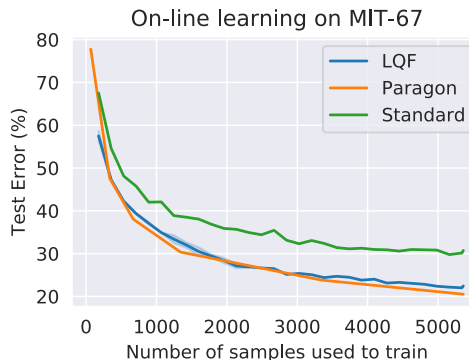

Figure 5: **Comparison of on-line learning methods.** Standard non-liner networks may converge to a bad local minimum when trained incrementally, and consequently have a sub-optimal performance compared to a non-linear network re-trained from scratch every time more data is obtained (paragon). On the other hand, LQF has a unique minimum to which it is guaranteed to converge, ensuring that the performance will be close to the paragon at all steps.

Figure 6: **Informativeness of samples. (Top row)** Seven representative examples from the 25 most informative images for a network trained on Oxford Flowers (complete set in appendix). The network considers more informative samples of flowers that are hard to distinguish, such as *English Marigold* and *Barbeton Daisy*, or *Hibiscus*, *Petunia* and *Pink Primrose*. On the other hand, images of flowers that have a distinctive shape (e.g., *Orchid*) or near duplicated images are not informative for the training **(bottom row)**. **(Right) Dataset summarization.** We plot the final test accuracy when training LQF after dropping the N most informative training examples (blue line) and the N less informative (orange line). The test performance decreases faster when dropping informative examples, as we would expect, validating our informativeness measure.

effective learning rates that yield the best performance. The interval of optimal ELRs for LQF is small, so one choice of ERL is nearly optimal for all tasks. Moreover, we do not observe significant changes in test accuracy for minor variations of ELR. On the other hand, for NLFT the ELR can vary by orders of magnitude depending on the task, and test performance can vary widely within that interval. Thus, fine-grained search of the ELR is necessary for NLFT but not for LQF. This robustness comes at the cost of a slight decrease in performance (Section 4.1, left), as expected due to the linearization.

5. Discussion

Usage of deep neural networks in practice often involves fine-tuning pre-trained models followed by heavy hyper-parameter optimization (HPO). This method can achieve good performance, but even minor variations of hyper-parameters can spoil performance. The low-data regime is also treacherous, as models can easily memorize the samples provided. It is difficult for non-expert to predict the effect of regularizers and hyper-parameters beyond trying it and seeing. Linear models, on the other hand, are supported by decades of theory that allows to predict the effect of changes in the data, or in the parameters, and bounds on the generalization error. However, their performance has not been comparable to that of black-box DNNs. LQF is a linear model that gets closer, to the point of making linearization viable beyond a theoretical analysis tool, as a practical method. The small drop in performance is offset by improved ease of training, interpretability, and cheaper hyper-parameter search.

LQF is not the solution to all fine-tuning problems. Being a first-order approximation, it can realistically be expected to work well when the task on which we wish to fine-tune

is sufficiently close to the task on which the model was pre-trained. In practice, this limitation may be circumvented by a "zoo" of different pretrained models, together with a selection technique to find the best initialization for the task [1]. Better understanding of the distance between tasks may also help characterizing the range in which linearization can be expected to work well.

Linearization can be made more flexible with architecture changes. For example, fine-grained classification tasks are typically far from generic models pre-trained on ImageNet. Indeed we observe that LQF drops in performance in fine-grained tasks, which have been noted to be further from ImageNet [29]. However, we have shown in Section 4.2, that for this specific case, simply replacing average pooling with bilinear pooling can stretch the performance of LQF in the fine-grained setting.

There are many possible uses for LQF, like any other linear model. Rather than providing an exhaustive list of all possibilities, we showed some illustrative examples. We leave various applications of LQF to future work, including its use in (i) uncertainty quantification, (ii) backward compatibility [44], (iii) leave-one-out cross validation without a validation set [9], (iv) active/online learning [45], (v) counterfactual analysis in the first order, (vi) continual learning [24], (vii) selective forgetting or machine unlearning [10, 12, 16], (viii) training including privacy or fairness constraints [7, 20], .

References

[1] Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless Fowlkes, Stefano Soatto, and Pietro Perona. Task2Vec: Task Embedding for Meta-Learning. *International Conference on Computer Vision*, 2019.

[2] Yu Bai, Ben Krause, Huan Wang, Caiming Xiong, and Richard Socher. Taylorized training: Towards better approximation of neural network training at finite width. *arXiv preprint arXiv:2002.04010*, 2020.

[3] Bjorn Barz and Joachim Denzler. Deep learning on small datasets without pre-training using cosine loss. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 1371–1380, 2020.

[4] Samyadeep Basu, Phil Pope, and Soheil Feizi. Influence functions in deep learning are fragile. In *International Conference on Learning Representations*, 2021.

[5] Imre Csiszar et al. Why least squares and maximum entropy? an axiomatic approach to inference for linear inverse problems. *The annals of statistics*, 19(4):2032–2066, 1991.

[6] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4109–4118, 2018.

[7] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[8] Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems*, pages 9550–9560, 2018.

[9] Ryan Giordano, William Stephenson, Runjing Liu, Michael Jordan, and Tamara Broderick. A swiss army infinitesimal jackknife. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1139–1147, 2019.

[10] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2019.

[11] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. In *Advances in Neural Information Processing Systems*, pages 10678–10688, 2019.

[12] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. *European Conference on Computer Vision*, 2020.

[13] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Interspeech*, volume 13, pages 1756–1760, 2013.

[14] G. Griffin, AD. Holub, and Pietro Perona. The caltech 256.

[15] Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016.

[16] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3832–3842. PMLR, 13–18 Jul 2020.

[17] Hrayr Harutyunyan, Alessandro Achille, Giovanni Paolini, Orchid Majumder, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Estimating informativeness of samples with smooth unique information. In *International Conference on Learning Representations*, 2021.

[18] Like Hui and Mikhail Belkin. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. In *International Conference on Learning Representations*, 2021.

[19] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[20] Michael Kearns and Aaron Roth. *The ethical algorithm: The science of socially aware algorithm design*. Oxford University Press, 2019.

[21] Daniel S Kermany, Michael Goldbaum, Wenjia Cai, Carolina CS Valentim, Huiying Liang, Sally L Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, et al. Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, 172(5):1122–1131, 2018.

[22] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA,*

USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[24] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[25] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[26] Simon Kornblith, Honglak Lee, Ting Chen, and Mohammad Norouzi. What's in a loss function for image classification? *arXiv preprint arXiv:2010.16402*, 2020.

[27] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[28] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[29] Hao Li, Pratik Chaudhari, Hao Yang, Michael Lam, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Rethinking the hyperparameters for fine-tuning. In *International Conference on Learning Representations*, 2020.

[30] Xuhong Li, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2825–2834, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[31] Tsung-Yu Lin and Subhransu Maji. Improved bilinear pooling with cnns. *arXiv preprint arXiv:1707.06772*, 2017.

[32] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 1449–1457, 2015.

[33] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning*, volume 30, page 3, 2013.

[34] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[35] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[36] Fangzhou Mu, Yingyu Liang, and Yin Li. Gradients as features for deep representation learning. In *International Conference on Learning Representations*, 2020.

[37] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning*, 2010.

[38] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[39] Cuong V Nguyen, Tal Hassner, Cedric Archambeau, and Matthias Seeger. Leep: A new measure to evaluate transferability of learned representations. *arXiv preprint arXiv:2002.12462*, 2020.

[40] Maria-Elena Nilsback and Andrew Zisserman. A visual vocabulary for flower classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1447–1454, 2006.

[41] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[42] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 413–420. IEEE, 2009.

[43] Sivaramakrishnan Rajaraman, Sameer K Antani, Mahdieh Poostchi, Kamolrat Silamut, Md A Hossain, Richard J Maude, Stefan Jaeger, and George R Thoma. Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. *PeerJ*, 6:e4568, 2018.

[44] Yantao Shen, Yuanjun Xiong, Wei Xia, and Stefano Soatto. Towards backward-compatible representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6368–6377, 2020.

[45] Alexander Strehl and Michael Littman. Online linear regression and its application to model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 20:1417–1424, 2007.

[46] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deeply learned face representations are sparse, selective, and robust. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2892–2900, 2015.

[47] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.