Supplementary Material *for* DLCFT: Deep Linear Continual Fine-Tuning for General Incremental Learning

A Second derivatives of softmax cross-entropy loss

Here, we show the vanishing behavior of the Hessian of the softmax cross-entropy loss. Let us consider a probabilistic classification model $p(y|x)$ where $x \in \mathbb{R}^M$ is the input to the softmax layer. The model is defined as,

$$p(y=k|x) = \frac{e^{x_k}}{\sum_m e^{x_m}}. \tag{15}$$

Assuming that the target label is t, the cross-entropy loss is,

$$\mathcal{L}(x) = -\log p(y=t|x) = -x_t + \log \sum_m e^{x_m}. \tag{16}$$

Then, the second derivatives of the softmax cross-entropy is,

$$\frac{\partial^2 \mathcal{L}(x)}{\partial x_i \partial x_j} = \frac{\partial^2}{\partial x_i \partial x_j}\left(-x_t + \log \sum_m e^{x_m}\right) \tag{17}$$

$$= \frac{\partial^2}{\partial x_i \partial x_j}\left(\log \sum_m e^{x_m}\right) \tag{18}$$

$$= \frac{\partial}{\partial x_j}\left(\frac{e^{x_i}}{\sum_m e^{x_m}}\right). \tag{19}$$

The diagonal entries of the Hessian where $j = i$ is,

$$\frac{\partial^2 \mathcal{L}(x)}{\partial x_i^2} = \frac{e^{x_i}\left(\sum_m e^{x_m} - e^{x_i}\right)}{\left(\sum_m e^{x_m}\right)^2} = \frac{\frac{e^{x_i}}{e^{x_t}}\left(1 + \sum_{m\neq t} \frac{e^{x_m}}{e^{x_t}} - \frac{e^{x_i}}{e^{x_t}}\right)}{\left(1 + \sum_{m\neq t} \frac{e^{x_m}}{e^{x_t}}\right)^2}. \tag{20}$$

And the off-diagonal entries where $j \neq i$ is,

$$\frac{\partial^2 \mathcal{L}(x)}{\partial x_i \partial x_j} = \frac{-e^{x_i} e^{x_j}}{\left(\sum_m e^{x_m}\right)^2} = \frac{-\frac{e^{x_i}}{e^{x_t}}\frac{e^{x_j}}{e^{x_t}}}{\left(1 + \sum_{m\neq t} \frac{e^{x_m}}{e^{x_t}}\right)^2}. \tag{21}$$

Now, let us consider the limiting case when the model converges to the target, $p(y=t|x) \to 1$, *i.e.*, if and only if $e^{x_m - x_t} \to 0$ for all $m \neq t$. Then, it can be seen from Eq. (20) and Eq. (21) that all the entries of the Hessian matrix converges to zero.

B Implementation of linearized neural network

Here, we provide implementation details for linearized nerual network. We consider a network with pre-trained parameter θ_0. We denote the neural network as $f(x;\theta_0)$. Then we apply first-order Taylor approximation with respect to the parameters to linearize the network around θ_0 as,

$$f(x;\theta_0+\Delta\theta) \simeq f(x;\theta_0) + D_\theta f(x;\theta_0)\cdot\Delta\theta, \tag{22}$$

where $D_\theta f(x;\theta_0)$ is the Jacobian of the network evaluated at (x,θ_0). For most neural networks, the Jacobian matrix is prohibitively expensive to compute and store due to the size of parameter dimension. To compute the forward pass, we use the modified forward pass method proposed in [1] based on forward-mode automatic differentiation algorithm, which efficiently computes Jacobian-vector products (JVP). Unlike backpropagation, the algorithm does not require extra memory footprint to compute the derivatives.

B.1 Augmented forward propagation

We implemented the modified forward pass by subclassing the layers in PyTorch [2] library.[1] The inputs and outputs of the custom layers are a tuple of hidden state and augmented state, such that the forward pass jointly computes the activations and the JVP.

Table 4 shows the formulas for the custom layer implementation. We use the same notation as [1] and denote the augmented state for JVP as $\partial_r x_l$. The JVP for full neural network is computed by feeding a zero-initialized vector having the same shape of the input as the incoming augmented input to the first layer. Finally, upon the completion of the forward pass, the computed hidden state and augmented state are summed to obtain the output of the linearized neural network.

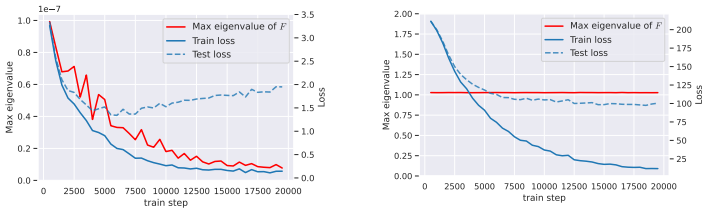
Table 4: Formulas for the augmented forward pass implementation. $\mathbb{1}\{x>0\}$ indicates a mask vector that has value 1 where the condition is satisfied and 0 otherwise. Max pooling layer is decomposed into MaxPoolIndices$(\cdot)$ and Gather$(\cdot)$ operations which gives the indices of the pooled values and aggregates the values using the indices.

Layer Type	Hidden State	Augmented State
Identity	x_l	$\partial_r x_l$
Linear, Conv	$f(x_l;W;b)$	$f(x_l;\Delta W,\Delta b)+f(\partial_r x_l;W,0)$
ReLU	$x_l\odot\mathbb{1}\{x_l>0\}$	$\partial_r x_l\odot\mathbb{1}\{x_l>0\}$
LeakyReLU	$x_l\odot\mathbb{1}\{x_l>0\}+\alpha x_l\odot\mathbb{1}\{x_l\le 0\}$	$\partial_r x_l\odot\mathbb{1}\{x_l>0\}+\alpha\partial_r x_l\odot\mathbb{1}\{x_l\le 0\}$
MaxPool	Gather$(x_l,\text{MaxPoolIndices}(x_l))$	Gather$(\partial_r x_l,\text{MaxPoolIndices}(x_l))$
AveragePool	AvgPool(x_l)	AvgPool$(\partial_r x_l)$

[1] https://github.com/pytorch/pytorch

C Additional experiments on vanishing curvature

Here, we show the curvature behavior during training using CIFAR-100 dataset.



(a) Curvature of SCE loss

(b) Curvature of MSE loss

Fig. 7: Maximum eigenvalue of the FIM during training using CIFAR-100 dataset.

D Comparison of curvature methods on data-IL

Here, we demonstrate the performance impact when combined with different curvature approximations. We highlight that the proposed method leads to performance gain in all types.

Table 5: Performance comparison of curvature methods on data-IL using Seq-CIFAR-100 with 10 tasks.

Curvature	Nonlinear+SCE	Linear+MSE
EWC	78.88	82.17
K-FAC	79.23	81.95
TK-FAC	-	82.70

E Backward transfer evaluations on task-IL

Here, we provide task-IL performance measured in Backward transfer (BWT) metric. Note that our method is on par with LwF and significantly outperforms other parameter regularization methods. Moreover, our method outperforms LwF in average accuracy (ACC) metric.

Table 6: Average accuracy (ACC) and backward transfer (BWT) on task-IL using Seq-CIFAR-100 with 10 tasks.

Method	ACC	BWT
LWF	92.16	0.04
EWC	77.44	-21.63
OSLA	81.03	-18.21
DLCFT (Ours)	95.79	-0.58

References

1. Mu, F., Liang, Y., Li, Y.: Gradients as features for deep representation learning. In: International Conference on Learning Representations (2020), https://openreview.net/forum?id=BkeoaeHKDS
2. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf