SIAM J. MATH. DATA SCI.
Vol. 2, No. 4, pp. 1096–1122

Active Subspace of Neural Networks: Structural Analysis and Universal Attacks[*]

Chunfeng Cui[†], Kaiqi Zhang[†], Talgat Daulbaev[‡], Julia Gusak[‡], Ivan Oseledets[§], and Zheng Zhang[†]

Abstract. Active subspace is a model reduction method widely used in the uncertainty quantification community. In this paper, we propose analyzing the internal structure and vulnerability of deep neural networks using active subspace. Firstly, we employ the active subspace to measure the number of "active neurons" at each intermediate layer, which indicates that the number of neurons can be reduced from several thousands to several dozens. This motivates us to change the network structure and to develop a new and more compact network, referred to as ASNet, that has significantly fewer model parameters. Secondly, we propose analyzing the vulnerability of a neural network using active subspace by finding an additive universal adversarial attack vector that can misclassify a dataset with a high probability. Our experiments on CIFAR-10 show that ASNet can achieve 23.98× parameter and 7.30× flops reduction. The universal active subspace attack vector can achieve around 20% higher attack ratio compared with the existing approaches in our numerical experiments. The PyTorch codes for this paper are available online.

Key words. active subspace, deep neural network, network reduction, universal adversarial perturbation

AMS subject classifications. 90C26, 15A18, 62G35

DOI. 10.1137/19M1296070

1. Introduction.

Deep neural networks have achieved impressive performance in many applications, such as computer vision [35], nature language processing [58], and speech recognition [23]. Most neural networks use deep structure (i.e., many layers) and a huge number of neurons to achieve a high accuracy and expressive power [19, 44]. However, it is still unclear how many layers and neurons are necessary. Employing an unnecessarily complicated deep neural network can cause huge extra costs in run-time and hardware resources. Driven by resource-constrained applications such as robotics and internet of things, there is an increasing interest in building smaller neural networks by removing network redundancy. Representative methods include network pruning and sharing [17, 25, 27, 39, 38], low-rank matrix and

[*]Received by the editors October 28, 2019; accepted for publication (in revised form) July 22, 2020; published electronically October 29, 2020.
https://doi.org/10.1137/19M1296070
Funding: The work of the first, second, and sixth authors was supported by a UCSB start-up grant. The work of the third, fourth, and fifth authors was supported by the Ministry of Education and Science of the Russian Federation (grant 14.756.31.0001).

[†]University of California Santa Barbara, Santa Barbara, CA 93106 USA (chunfengcui@ucsb.edu, kzhang07@ucsb.edu, zhengzhang@ece.ucsb.edu).

[‡]Skolkovo Institute of Science and Technology, Moscow, Russia (talgat.daulbaev@skoltech.ru, y.gusak@skoltech.ru).

[§]Skolkovo Institute of Science and Technology and Institute of Numerical Mathematics of Russian Academy of Sciences, Moscow, Russia (ivan.oseledets@gamil.com).

tensor factorization [18, 26, 36, 43, 49], parameter quantization [12, 15], knowledge distillation [28, 46], and so forth. However, most existing methods delete model parameters directly without changing the network architecture [7, 25, 27, 38].

Another important issue of deep neural networks is the lack of robustness. A deep neural network is desired to maintain good performance for noisy or corrupted data to be deployed in safety-critical applications such as autonomous driving and medical image analysis. However, recent studies have revealed that many state-of-the-art deep neural networks are vulnerable to small perturbations [54]. A substantial number of methods have been proposed to generate adversarial examples. Representative works can be classified into four classes [52], including optimization methods [8, 41, 40, 54], sensitive features [22, 45], geometric transformations [16, 32], and generative models [4]. However, these methods share a fundamental limitation: each perturbation is designed for a given data point, and one has to implement the algorithm again to generate the perturbation for a new data sample. Recently, several methods have also been proposed to compute a universal adversarial attack to fool a dataset simultaneously (rather than one data sample) in various applications, such as computer vision [40], speech recognition [42], audio [1], and text classifier [5]. However, all the above methods only solve a series of data-dependent subproblems. Recently, Khrulkov and Oseledets [33] proposed constructing universal perturbation by computing the (p, q)-singular vectors of the Jacobian matrices of hidden layers of a network.

This paper investigates the above two issues with the active subspace method [9, 10, 48] that was originally developed for uncertainty quantification. The key idea of the active subspace is to identify the low-dimensional subspace constructed by some important directions that can contribute significantly to the variance of the multivariable function. These directions correspond to the principal components of the uncentered covariance matrix of gradients. Afterwards, a response surface can be constructed in this low-dimensional subspace to reduce the number of parameters for partial differential equations [10] and uncertainty quantification [11]. However, the power of active subspace in analyzing and attacking deep neural networks has not been explored.

1.1. Paper contributions.

The contribution of this manuscript is twofold.

- Firstly, we apply the active subspace to some intermediate layers of a deep neural network and try to answer the following question: *How many neurons and layers are important in a deep neural network?* Based on the active subspace, we propose the definition of "active neurons." Figure 1(a) shows that even though there are tens of thousands of neurons, only dozens of them are important from the active subspace point of view. Figure 1(b) further shows that most of the neural network parameters are distributed in the last few layers. This motivates us to cut off the tail layers and replace them with a smaller and simpler new framework called ASNet. ASNet contains three parts: the first few layers of a deep neural network, an active subspace layer that maps the intermediate neurons to a low-dimensional subspace, and a polynomial chaos expansion layer that projects the reduced variables to the outputs. Our numerical experiments show that the proposed ASNet has far fewer model parameters than the original one. ASNet can also be combined with existing structured re-training methods (e.g., pruning and quantization) to get better accuracy while using fewer model parameters.

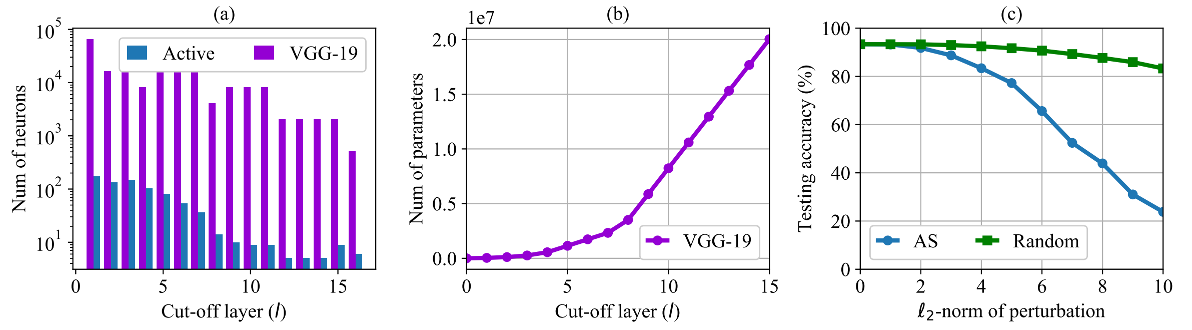

Figure 1. *Structural analysis of deep neural networks by the active subspace (AS). All experiments are conducted on CIFAR-*10 *by VGG-*19*.* (a) *The number of neurons can be significantly reduced by the active subspace. Here, the number of active neurons is defined by Definition* 3.1 *with a threshold* $\epsilon = 0.05$. (b) *Most of the parameters are distributed in the last few layers.* (c) *The active subspace direction can perturb the network significantly.*

- Secondly, we use active subspace to develop a new universal attack method to fool deep neural networks on a whole dataset. We formulate this problem as a ball-constrained loss maximization problem and propose a heuristic projected gradient descent algorithm to solve it. At each iteration, the ascent direction is the dominant active subspace, and the stepsize is decided by the backtracking algorithm. Figure 1(c) shows that the attack ratio of the active subspace direction is much higher than that of the random vector.

The rest of this manuscript is organized as follows. In section 2, we review the key idea of active subspace. Based on the active subspace method, section 3 shows how to find the number of active neurons in a deep neural network and further proposes a new and compact network, referred to as ASNet. Section 4 develops a new universal adversarial attack method based on active subspace. The numerical experiments for both ASNet and universal adversarial attacks are presented in section 5. Finally, we conclude this paper in section 6.

2. Active subspace.

Active subspace is an efficient tool for functional analysis and dimension reduction. Its key idea is to construct a low-dimensional subspace for the input variables in which the function value changes dramatically. Given a continuous function $c(\mathbf{x})$ with $\mathbf{x}$ described by the probability density function $\rho(\mathbf{x})$, one can construct an uncentered covariance matrix for the gradient: $\mathbf{C} = \mathbb{E}[\nabla c(\mathbf{x})\nabla c(\mathbf{x})^T]$. Suppose the matrix $\mathbf{C}$ admits the following eigenvalue decomposition:

$$\mathbf{C} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T, \tag{2.1}$$

where $\mathbf{V}$ includes all orthogonal eigenvectors and

$$\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n), \ \ \lambda_1 \geq \cdots \geq \lambda_n \geq 0 \tag{2.2}$$

are the eigenvalues. All the eigenvalues are nonnegative because $\mathbf{C}$ is positive semidefinite. One can split the matrix $\mathbf{V}$ into two parts,

$$\mathbf{V} = [\mathbf{V}_1, \ \mathbf{V}_2], \ \text{ where } \mathbf{V}_1 \in \mathbb{R}^{n\times r} \text{ and } \mathbf{V}_2 \in \mathbb{R}^{n\times(n-r)}. \tag{2.3}$$

The subspace spanned by matrix $\mathbf{V}_1 \in \mathbb{R}^{n\times r}$ is called an active subspace [48], because $c(\mathbf{x})$ is sensitive to perturbation vectors inside this subspace.

Remark 2.1 (relationships with the principal component analysis). Given a set of data $\mathbf{X} = [\mathbf{x}^1, \dots, \mathbf{x}^m]$ with each column representing a data sample and where each row is zero-mean, the first principal component $\mathbf{w}_1$ inherits the maximal variance from $\mathbf{X}$, namely,

$$\mathbf{w}_1 = \underset{\|\mathbf{w}\|_2=1}{\operatorname{argmax}} \sum_{i=1}^{m} (\mathbf{w}_1^T \mathbf{x}^i)^2 = \underset{\|\mathbf{w}\|_2=1}{\operatorname{argmax}} \ \mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w}. \tag{2.4}$$

The variance is maximized when $\mathbf{w}_1$ is the eigenvector associated with the largest eigenvalue of $\mathbf{X}\mathbf{X}^T$. The first r principal components are the r eigenvectors associated with the r largest eigenvalues of $\mathbf{X}\mathbf{X}^T$. The main difference with the active subspace is that the principal component analysis uses the covariance matrix of input dataset $\mathbf{X}$, but the active subspace method uses the covariance matrix of gradient $\nabla c(\mathbf{x})$. Hence, a perturbation along the direction $\mathbf{w}_1$ from (2.4) only guarantees the variability in the data and does not necessarily cause a significant change on the value of $c(\mathbf{x})$.

The following lemma quantitatively describes that $c(\mathbf{x})$ varies more on average along the directions defined by the columns of $\mathbf{V}_1$ than the directions defined by the columns of $\mathbf{V}_2$.

Lemma 2.2 (see [10]). *Suppose $c(\mathbf{x})$ is a continuous function and $\mathbf{C}$ is obtained from (2.1). For the matrices $\mathbf{V}_1$ and $\mathbf{V}_2$ generated by (2.3), and the reduced vector*

$$\mathbf{z} = \mathbf{V}_1^T\mathbf{x} \ \textit{and} \ \tilde{\mathbf{z}} = \mathbf{V}_2^T\mathbf{x}, \tag{2.5}$$

it holds that

$$\begin{aligned} &\mathbb{E}_{\mathbf{x}}[\nabla_{\mathbf{z}} c(\mathbf{x})^T \nabla_{\mathbf{z}} c(\mathbf{x})] = \lambda_1 + \cdots + \lambda_r, \\ &\mathbb{E}_{\mathbf{x}}[\nabla_{\tilde{\mathbf{z}}} c(\mathbf{x})^T \nabla_{\tilde{\mathbf{z}}} c(\mathbf{x})] = \lambda_{r+1} + \cdots + \lambda_n. \end{aligned} \tag{2.6}$$

Sketch of proof [10].

$$\begin{aligned} &\mathbb{E}_{\mathbf{x}}[\nabla_{\mathbf{z}} c(\mathbf{x})^T \nabla_{\mathbf{z}} c(\mathbf{x})] \\ &\quad = \operatorname{trace}\left(\mathbb{E}_{\mathbf{x}}[\nabla_{\mathbf{z}} c(\mathbf{x}) \nabla_{\mathbf{z}} c(\mathbf{x})^T]\right) \\ &\quad = \operatorname{trace}\left(\mathbb{E}_{\mathbf{x}}[\mathbf{V}_1^T \nabla_{\mathbf{x}} c(\mathbf{x}) \nabla_{\mathbf{x}} c(\mathbf{x})^T \mathbf{V}_1]\right) \\ &\quad = \operatorname{trace}\left(\mathbf{V}_1^T\mathbf{C}\mathbf{V}_1\right) \\ &\quad = \lambda_1 + \cdots + \lambda_r. \end{aligned}$$

■

When $\lambda_{r+1} = \cdots = \lambda_n = 0$, Lemma 2.2 implies $\nabla_{\tilde{\mathbf{z}}} c(\mathbf{x})$ is zero everywhere, i.e., $c(\mathbf{x})$ is $\tilde{z}$-invariant. In this case, we may reduce $\mathbf{x} \in \mathbb{R}^n$ to a low-dimensional vector $\mathbf{z} = \mathbf{V}_1^T\mathbf{x} \in \mathbb{R}^r$ and construct a new response surface $g(\mathbf{z})$ to represent $c(\mathbf{x})$. Otherwise, if λ_{r+1} is small, we may still construct a response surface $g(\mathbf{z})$ to approximate $c(\mathbf{x})$ with a bounded error, as shown in the following lemma.

2.1. Response surface. For a fixed $\mathbf{z}$, the best guess for g is the conditional expectation of c given $\mathbf{z}$, i.e.,

$$g(\mathbf{z}) = \mathbb{E}_{\tilde{\mathbf{z}}}[c(\mathbf{x})|\mathbf{z}] = \int c(\mathbf{V}_1\mathbf{z} + \mathbf{V}_2\tilde{\mathbf{z}})\rho(\tilde{\mathbf{z}}|\mathbf{z})d\tilde{\mathbf{z}}. \tag{2.7}$$

Based on the Poincaré inequality, the following approximation error bound is obtained [10].

Lemma 2.3. *Assume that $c(\mathbf{x})$ is absolutely continuous and square integrable with respect to the probability density function $\rho(\mathbf{x})$; then the approximation function $g(\mathbf{z})$ in (2.7) satisfies*

$$\mathbb{E}[(c(\mathbf{x}) - g(\mathbf{z}))^2] \leq O(\lambda_{r+1} + \cdots + \lambda_n). \tag{2.8}$$

Sketch of proof [10].

$$\begin{aligned}
&\mathbb{E}_{\mathbf{x}}[(c(\mathbf{x}) - g(\mathbf{z}))^2] \\
&\quad = \mathbb{E}_{\mathbf{z}}[\mathbb{E}_{\tilde{\mathbf{z}}}[(c(\mathbf{x}) - g(\mathbf{z}))^2 \,|\mathbf{z}]] \\
&\quad \leq \text{const} \times \mathbb{E}_{\mathbf{z}}\left[\mathbb{E}_{\tilde{\mathbf{z}}}\left[\nabla_{\tilde{\mathbf{z}}} c(\mathbf{x})^T \nabla_{\tilde{\mathbf{z}}} c(\mathbf{x})|\mathbf{z}\right]\right] \quad \text{(Poincaré inequality)} \\
&\quad = \text{const} \times \mathbb{E}_{\mathbf{x}}\left[\nabla_{\tilde{\mathbf{z}}} c(\mathbf{x})^T \nabla_{\tilde{\mathbf{z}}} c(\mathbf{x})\right] \\
&\quad = \text{const} \times (\lambda_{r+1} + \cdots + \lambda_n) \quad \text{(Lemma 2.2)} \\
&\quad = O(\lambda_{r+1} + \cdots + \lambda_n).
\end{aligned}$$

■

In other words, the active subspace approximation error will be small if $\lambda_{r+1}, \ldots, \lambda_n$ are negligible.

3. Active subspace for structural analysis and compression of deep neural networks.

This section applies the active subspace to analyze the internal layers of a deep neural network to reveal the number of important neurons at each layer. Afterward, a new network called ASNet is built to reduce the storage and computational complexity.

3.1. Deep neural networks. A deep neural network can be described as

$$f(\mathbf{x}_0) = f_L\left(f_{L-1} \ldots (f_1(\mathbf{x}_0))\right), \tag{3.1}$$

where $\mathbf{x}_0 \in \mathbb{R}^{n_0}$ is an input, L is the total number of layers, and $f_l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$ is a function representing the lth layer (e.g., combinations of convolution or fully connected, batch normalization, rectified linear unit (ReLU), or pooling layers). For any $1 \leq l \leq L$, we rewrite the above feed-forward model as a superposition of functions, i.e.,

$$f(\mathbf{x}_0) = f^l_{\text{post}}(f^l_{\text{pre}}(\mathbf{x}_0)), \tag{3.2}$$

where the *pre-model* $f^l_{\text{pre}}(\cdot) = f_l \ldots (f_1(\cdot))$ denotes all operations before the lth layer and the *post-model* $f^l_{\text{post}}(\cdot) = f_L \ldots (f_{l+1}(\cdot))$ denotes all succeeding operations. The intermediate neuron $\mathbf{x}_l = f^l_{\text{pre}}(\mathbf{x}_0) \in \mathbb{R}^{n_l}$ usually lies in a high dimension. We aim to study whether such a high dimensionality is necessary. If not, how can we reduce it?

3.2. The number of active neurons.

Denote loss$(\cdot)$ as the loss function, and

$$c_l(\mathbf{x}) = \text{loss}(f^l_{\text{post}}(\mathbf{x})). \tag{3.3}$$

The covariance matrix $\mathbf{C} = \mathbb{E}[\nabla c_l(\mathbf{x})\nabla c_l(\mathbf{x})^T]$ admits the eigenvalue decomposition $\mathbf{C} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ with $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_{n_l})$. We try to extract the active subspace of $c_l(\mathbf{x})$ and reduce the intermediate vector $\mathbf{x}$ to a low dimension. Here the intermediate neuron $\mathbf{x}$, the covariance matrix $\mathbf{C}$, eigenvalues $\mathbf{\Lambda}$, and eigenvectors $\mathbf{V}$ are also related to the layer index l, but we ignore the index for simplicity.

Definition 3.1. *Suppose $\mathbf{\Lambda}$ is computed by (2.2). For any layer index $1 \le l \le L$, we define the number of active neurons $n_{l,AS}$ as follows:*

$$n_{l,AS} = \arg\min \left\{ i : \frac{\lambda_1 + \ldots + \lambda_i}{\lambda_1 + \ldots + \lambda_{n_l}} \ge 1 - \epsilon \right\}, \tag{3.4}$$

where $\epsilon > 0$ is a user-defined threshold.

Based on Definition 3.1, the post-model can be approximated by an $n_{l,\text{AS}}$-dimensional function with a high accuracy, i.e.,

$$g_l(\mathbf{z}) = \mathbb{E}_{\tilde{\mathbf{z}}}[c_l(\mathbf{x})|\mathbf{z}]. \tag{3.5}$$

Here $\mathbf{z} = \mathbf{V}_1^T\mathbf{x} \in \mathbb{R}^{n_{l,AS}}$ plays the role of active neurons, $\tilde{\mathbf{z}} = \mathbf{V}_2^T\mathbf{x} \in \mathbb{R}^{n - n_{l,AS}}$, and $\mathbf{V} = [\mathbf{V}_1, \mathbf{V}_2]$.

Lemma 3.1. *Suppose the input $\mathbf{x}_0$ is bounded. Consider a deep neural network with the following operations: convolution, fully connected, ReLU, batch normalization, max pooling, and equipped with the cross entropy loss function. Then for any $l \in \{1, \ldots, L\}$, $\mathbf{x} = f^l_{pre}(\mathbf{x}_0)$, and $c_l(\mathbf{x}) = loss(f^l_{post}(\mathbf{x}))$, the $n_{l,AS}$-dimensional function $g_l(\mathbf{z})$ defined in (3.5) satisfies*

$$\mathbb{E}_{\mathbf{z}}\left[(g_l(\mathbf{z}))^2\right] \le 2\mathbb{E}_{\mathbf{x}_0}\left[(c_0(\mathbf{x_0}))^2\right] + O(\epsilon). \tag{3.6}$$

Proof. Denote $c_l(\mathbf{x}) = \text{loss}(f_L(\ldots(f_{l+1}(\mathbf{x})))$, where $\text{loss}(\mathbf{y}) = -\log(\exp(y_b))/(\sum_{i=1}^{n_L} \exp(y_i))$ is the cross entropy loss function, b is the true label, and n_L is the total number of classes. We first show $c_l(\mathbf{x})$ is absolutely continuous and square integrable, and then apply Lemma 2.3 to derive (3.6).

Firstly, all components of $c_l(\mathbf{x})$ are Lipschitz continuous because (1) the convolution, fully connected, and batch normalization operations are all linear; (2) the max pooling and ReLU functions are nonexpansive. Here, a mapping m is nonexpansive if $\|m(\mathbf{x}) - m(\mathbf{y})\| \le \|\mathbf{x} - \mathbf{y}\|$; (3) the cross entropy loss function is smooth with an upper bounded gradient, i.e., $\|\nabla\text{loss}(\mathbf{y})\| = \|\mathbf{e}_b - \exp(\mathbf{y})/\sum_{i=1}^{n_L}\exp(y_i)\| \le \sqrt{n_L}$. The composition of two Lipschitz continuous functions is also Lipschitz continuous: suppose the Lipschitz constants for f_1 and f_2 are α_1 and α_2, respectively; it holds that $\|f_1(f_2(\bar{\mathbf{x}})) - f_1(f_2(\underline{\mathbf{x}}))\| \le \alpha_1\|f_2(\bar{\mathbf{x}}) - f_2(\underline{\mathbf{x}})\| \le \alpha_1\alpha_2\|\bar{\mathbf{x}} - \underline{\mathbf{x}}\|$ for any vectors $\bar{\mathbf{x}}$ and $\underline{\mathbf{x}}$. By recursively applying the above rule, $c_l(\mathbf{x})$ is Lipschitz continuous:

$$\begin{aligned}\|c_l(\bar{\mathbf{x}}) - c_l(\underline{\mathbf{x}})\|_2 &= \|\text{loss}(f_L(\dots(f_{l+1}(\bar{\mathbf{x}})))) - \text{loss}(f_L(\dots(f_{l+1}(\underline{\mathbf{x}}))))\|_2 \\ &\le \sqrt{n_L}\alpha_L \dots \alpha_{l+1}\|\bar{\mathbf{x}} - \underline{\mathbf{x}}\|_2.\end{aligned}$$

The intermediate neuron $\mathbf{x}$ is in a bounded domain because the input $\mathbf{x}_0$ is bounded and all functions $f_i(\cdot)$ are either continuous or nonexpansive. Based on the fact that any Lipschitz-continuous function is also absolutely continuous on a compact domain [47], we conclude that $c_l(\mathbf{x})$ is absolutely continuous.

Secondly, because $\mathbf{x}$ is bounded and $c_l(\mathbf{x})$ is continuous, both $c_l(\mathbf{x})$ and its square integral will be bounded, i.e., $\int (c_l(\mathbf{x})^2\rho(\mathbf{x})d\mathbf{x} < \infty$.

Finally, by Lemma 2.3, it holds that

$$\mathbb{E}_{\mathbf{x}}[(c_l(\mathbf{x}) - g_l(\mathbf{z}))^2] \le O(\lambda_{n_{l,AS}+1} + \cdots + \lambda_n).$$

From Definition 3.1, we have

$$\lambda_{n_{l,AS}+1} + \cdots + \lambda_n \le (\lambda_1 + \cdots + \lambda_n)\epsilon = \|\mathbf{C}^{1/2}\|_F^2\epsilon = O(\epsilon).$$

The last equality uses that $\|\mathbf{C}^{1/2}\|_F$ is upper bounded because $c_l(\mathbf{x})$ is Lipschitz continuous with a bounded gradient. Consequently, we have

$$\begin{aligned}&\mathbb{E}_{\mathbf{x}}[(g_l(\mathbf{z}))^2] \\ &\quad = \mathbb{E}_{\mathbf{x}}[(g_l(\mathbf{z}) - c_l(\mathbf{x}) + c_l(\mathbf{x}))^2] \\ &\quad \le 2\mathbb{E}_{\mathbf{x}}[(c_l(\mathbf{x}))^2] + 2\mathbb{E}_{\mathbf{x}}[(c_l(\mathbf{x}) - g_l(\mathbf{z}))^2] \\ &\quad = 2\mathbb{E}_{\mathbf{x}_0}[(c_0(\mathbf{x_0}))^2] + 2\mathbb{E}_{\mathbf{x}}[(c_l(\mathbf{x}) - g_l(\mathbf{z}))^2] \\ &\quad \le 2\mathbb{E}_{\mathbf{x}_0}[(c_0(\mathbf{x_0}))^2] + O(\epsilon).\end{aligned}$$

The proof is completed. ■

The above lemma shows that the active subspace method can reduce the number of neurons of the lth layer from n_l to $n_{l,AS}$. The loss for the low-dimensional function $g_l(\mathbf{z})$ is bounded by two terms: the loss $c_0(\mathbf{x}_0)$ of the original network, and the threshold ϵ related to $n_{l,AS}$. Although the loss function is the cross entropy loss, not the classification error, it is believed that a small loss will result in a small classification error. Further, the result in Lemma 3.1 is valid for thr fixed parameters in the pre-model. In practice, we can fine-tune the pre-model to achieve better accuracy.

Further, as stated in the following subsection, a small number of active neurons $n_{l,AS}$ is critical to get a high compress ratio. From Definition 3.1, $n_{l,AS}$ depends on the eigenvalue distribution of the covariance matrix $\mathbf{C}$. For a proper network structure and a good choice of the layer index l, if the eigenvalues of $\mathbf{C}$ are dominated by the first few eigenvalues, then $n_{l,AS}$ will be small. For instance, in Figure 5(a), the eigenvalues for layers $4 \le l \le 7$ of VGG-19 are nearly exponential decreasing to zero.

3.3. Active subspace network (ASNet). This subsection proposes a new network called ASNet that can reduce both the storage and computational cost. Given a deep neural network, we first choose a proper layer l and project the high-dimensional intermediate neurons to a low-dimensional vector in the active subspace. Afterward, the post-model is deleted completely

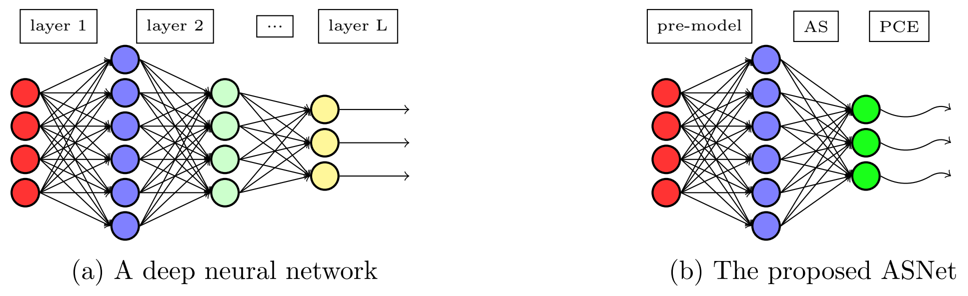

Figure 2. (a) *The original deep neural network.* (b) *The proposed ASNet with three parts: a pre-model, an active subspace (AS) layer, and a polynomial chaos expansion (PCE) layer.*

and replaced with a nonlinear model that maps the low-dimensional active feature vector to the output directly. This new network, called ASNet, has three parts:

(1) Pre-model: the pre-model includes the first l layers of a deep neural network.
(2) Active subspace layer: a linear projection from the intermediate neurons to the low-dimensional active subspace.
(3) Polynomial chaos expansion layer: the polynomial chaos expansion [20, 56] maps the active subspace variables to the output.

The initialization for the active subspace layer and polynomial chaos expansion layer are presented in sections 3.4 and 3.5, respectively. We can also retrain all the parameters to increase the accuracy. The whole procedure is illustrated in Figure 2(b) and Algorithm 3.1.

Algorithm 3.1. The training procedure of the ASNet.

Input: A pretrained deep neural network, the layer index l, and the number of active neurons r.

Step 1 **Initialize the active subspace layer.** The active subspace layer is a linear projection where the projection matrix $\mathbf{V}_1 \in \mathbb{R}^{n\times r}$ is computed by Algorithm 3.2. If r is not given, we use $r = n_{\text{AS}}$ defined in (3.4) by default.

Step 2 **Initialize the polynomial chaos expansion layer.** The polynomial chaos expansion layer is a nonlinear mapping from the reduced active subspace to the outputs, as shown in (3.10). The weights $\mathbf{c}_{\boldsymbol{\alpha}}$ is computed by (3.12).

Step 3 **Construct the ASNet.** Combine the pre-model (the first l layers of the deep neural network) with the active subspace and polynomial chaos expansion layers as a new network, referred to as ASNet.

Step 4 **Fine-tuning.** Retrain all the parameters in pre-model, active subspace layer, and polynomial chaos expansion layer in ASNet for several epochs by a proper (variant of) the stochastic gradient descent algorithm.

Output: A new network ASNet

3.4. The active subspace layer.

This subsection presents an efficient method to project the high-dimensional neurons to the active subspace. Given a dataset $\mathcal{D} = \{\mathbf{x}^1, \dots, \mathbf{x}^m\}$, the empirical covariance matrix is computed by $\hat{\mathbf{C}} = \frac{1}{m}\sum_{i=1}^m \nabla c_l(\mathbf{x}^i)\nabla c_l(\mathbf{x}^i)^T$. When ReLU is applied as an activation, $c_l(\mathbf{x})$ is not differentiable. In this case, we use ∇ to denote the subgradient with a little abuse of notation.

Instead of calculating the eigenvalue decomposition of $\hat{\mathbf{C}}$, we compute the singular value decomposition of $\hat{\mathbf{G}}$ to save the computation cost:

$$\hat{\mathbf{G}} = [\nabla c_l(\mathbf{x}^1), \ldots, \nabla c_l(\mathbf{x}^m)] = \hat{\mathbf{V}}\hat{\boldsymbol{\Sigma}}\hat{\mathbf{U}}^T \in \mathbb{R}^{n_l \times m} \text{ with } \hat{\boldsymbol{\Sigma}} = \operatorname{diag}(\hat{\sigma}_1, \ldots, \hat{\sigma}_{n_l}). \tag{3.7}$$

Here, the eigenvectors $\mathbf{V}$ are approximated by the left singular vectors $\hat{\mathbf{V}}$, and the eigenvalues $\boldsymbol{\Lambda}$ are approximated by the singular values of $\hat{\mathbf{G}}$, i.e., $\boldsymbol{\Lambda} \approx \hat{\boldsymbol{\Sigma}}^2$.

We use the memory-saving frequent direction method [21] to compute the r dominant singular value components, i.e., $\hat{\mathbf{G}} \approx \hat{\mathbf{V}}_r\hat{\boldsymbol{\Sigma}}_r\hat{\mathbf{U}}_r^T$. Here r is smaller than the total number of samples. The frequent direction approach only stores an $n \times r$ matrix $\mathbf{S}$. At the beginning, each column of $\mathbf{S} \in \mathbb{R}^{n\times r}$ is initialized by a gradient vector. Then the randomized singular value decomposition [24] is used to generate $\mathbf{S} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$. Afterwards, $\mathbf{S}$ is updated in the following way:

$$\mathbf{S} \leftarrow \mathbf{V}\sqrt{\boldsymbol{\Sigma}^2 - \sigma_r^2}. \tag{3.8}$$

Now the last column of $\mathbf{S}$ is zero, and we replace it with the gradient vector of a new sample. By repeating this process, $\mathbf{S}\mathbf{S}^T$ will approximate $\hat{\mathbf{G}}\hat{\mathbf{G}}^T$ with a high accuracy, and $\mathbf{V}$ will approximate the left singular vectors of $\hat{\mathbf{G}}$. The algorithm framework is presented in Algorithm 3.2.

Algorithm 3.2. The frequent direction algorithm for computing the active subspace.

Input: A dataset with m_{AS} input samples $\{\mathbf{x}_0^j\}_{j=1}^{m_{AS}}$, a pre-model $f_{\text{pre}}^l(\cdot)$, a subroutine for computing $\nabla c_l(\mathbf{x})$, and the dimension of truncated singular value decomposition r.

```
1: Select $r$ samples $\mathbf{x}_0^i$, compute $\mathbf{x}^i = f_{\text{pre}}^l(\mathbf{x}_0^i)$, and construct an initial matrix $\mathbf{S} \leftarrow [\nabla c_l(\mathbf{x}^1), \ldots, \nabla c_l(\mathbf{x}^r)]$.
2: for $t$=1, 2, ..., do
3:   Compute the singular value decomposition $\mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T \leftarrow \operatorname{svd}(\mathbf{S})$, where $\boldsymbol{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$.
4:   If the maximal number of samples $m_{AS}$ is reached, stop.
5:   Update $\mathbf{S}$ by the soft-thresholding (3.8).
6:   Get a new sample $\mathbf{x}_0^{\text{new}}$, compute $\mathbf{x}^{\text{new}} = f_{\text{pre}}^l(\mathbf{x}_0^{\text{new}})$, and replace the last column of $\mathbf{S}$ (now all zeros) by the gradient vector $\mathbf{S}(:, r) \leftarrow \nabla c_l(\mathbf{x}^{\text{new}})$.
7: end for
   Output: The projection matrix $\mathbf{V} \in \mathbb{R}^{n_l\times r}$ and the singular values $\boldsymbol{\Sigma} \in \mathbb{R}^{r\times r}$.
```

After obtaining $\boldsymbol{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$, we can approximate the number of active neurons as

$$\hat{n}_{l,AS} = \arg\min \left\{ i : \quad \frac{\sqrt{\sigma_1^2 + \cdots + \sigma_i^2}}{\sqrt{\sigma_1^2 + \cdots + \sigma_r^2}} \geq 1 - \epsilon \right\}. \tag{3.9}$$

Under the condition that $\sigma_i^2 \to \lambda_i$ for $i = 1, \ldots, r$ and $\lambda_i \to 0$ for $i = r+1, \ldots, n_l$, (3.9) can approximate $n_{l,AS}$ in (3.4) with a high accuracy. Further, the projection matrix $\hat{\mathbf{V}}_1$ is chosen as the first $\hat{n}_{l,AS}$ columns of $\mathbf{V}$. Consequently, the numerical cost for computing the projection matrix and eigenvalues is reduced. Specifically, the storage cost is reduced from $O(n_l^2)$ to $O(n_l r)$ and the computational cost is reduced from $O(n_l^2 r)$ to $O(n_l r^2)$, respectively.

3.5. Polynomial chaos expansion layer. We continue to construct a new surrogate model to approximate the post-model of a deep neural network. This problem can be regarded as an uncertainty quantification problem if we set $\mathbf{z}$ as a random vector. We choose the nonlinear polynomial because it has higher expressive power than linear functions.

By the polynomial chaos expansion [55], the network output $\mathbf{y} \in \mathbb{R}^{n_L}$ is approximated by a linear combination of the orthogonal polynomial basis functions:

$$\hat{\mathbf{y}} \approx \sum_{|\boldsymbol{\alpha}|=0}^{p} \mathbf{c}_{\boldsymbol{\alpha}} \boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z}), \text{ where } |\boldsymbol{\alpha}| = \alpha_1 + \cdots + \alpha_d. \tag{3.10}$$

Here $\boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z})$ is a multivariate polynomial basis function chosen based on the probability density function of $\mathbf{z}$. When the parameters $\mathbf{z} = [z_1, \ldots, z_r]^T$ are independent, both the joint density function and the multivariable basis function can be decomposed into products of one-dimensional functions, i.e., $\rho(\mathbf{z}) = \rho_1(z_1) \ldots \rho_r(z_r)$, $\boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z}) = \phi_{\alpha_1}(z_1)\phi_{\alpha_2}(z_2) \ldots \phi_{\alpha_r}(z_r)$. The marginal basis function $\phi_{\alpha_j}(z_j)$ is uniquely determined by the marginal density function $\rho_j(z_j)$. The scatter plot in Figure 3 shows that the marginal probability density of z_i is close to a Gaussian distribution.

Suppose $\rho_j(z_j)$ follows a Gaussian distribution; then $\phi_{\alpha_j}(z_j)$ will be a Hermite polynomial [37], i.e.,

$$\phi_0(z) = 1, \; \phi_1(z) = z, \; \phi_2(z) = 4z^2 - 2, \; \phi_{p+1}(z) = 2z\phi_p(z) - 2p\phi_{p-1}(z). \tag{3.11}$$

In general, the elements in $\mathbf{z}$ can be non-Gaussian correlated. In this case, the basis functions $\{\boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z})\}$ can be built via the Gram–Schmidt approach described in [13].

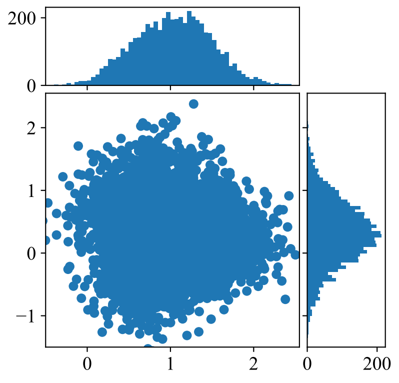

Figure 3. *Distribution of the first two active subspace variables at the 6th layer of VGG-19 for CIFAR-10.*

The coefficient $\mathbf{c}_{\boldsymbol{\alpha}}$ can be computed by a linear least-square optimization. Denote $\mathbf{z}^j = \hat{\mathbf{V}}_1^T f_{\text{pre}}^l(\mathbf{x}_0^j)$ as the random samples and $\mathbf{y}^j$ as the network output for $j = 1, \ldots, m_{\text{PCE}}$. The coefficient vector $\mathbf{c}_{\boldsymbol{\alpha}}$ can be computed by

$$\min_{\{\mathbf{c}_{\boldsymbol{\alpha}}\}} \quad \frac{1}{m_{\text{PCE}}} \sum_{j=1}^{m_{\text{PCE}}} \|\mathbf{y}^j - \sum_{|\boldsymbol{\alpha}|=0}^{p} \mathbf{c}_{\boldsymbol{\alpha}} \boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z}^j)\|^2. \tag{3.12}$$

Based on the Nyquist–Shannon sampling theorem, the number of samples to train $\mathbf{c}_{\boldsymbol{\alpha}}$ needs to satisfy $m_{\text{PCE}} \geq 2n_{\text{basis}} = 2\binom{r+p}{p}$. However, this number can be reduced to a smaller set of "important" samples by the D-optimal design [59] or the sparse regularization approach [14].

The polynomial chaos expansion builds a surrogate model to approximate the deep neural network output $\mathbf{y}$. This idea is similar to the knowledge distillation [28], where a pretrained teacher network teaches a smaller student network to learn the output feature. However, our polynomial chaos layer uses one nonlinear projection, whereas the knowledge distillation uses a series of layers. Therefore, the polynomial chaos expansion is more efficient in terms of computational and storage cost. The polynomial chaos expansion layer is also different from the polynomial activation because the dimension of $\mathbf{z}$ may be different from that of output $\mathbf{y}$.

The problem (3.12) is convex, and any first order method can converge to a global optimal solution. Denote the optimal coefficients as $\mathbf{c}_\alpha^*$ and the finial objective value as δ^*, i.e.,

$$\delta^* = \frac{1}{m_{\text{PCE}}} \sum_{j=1}^{m_{\text{PCE}}} \|\mathbf{y}^j - \psi^*(\mathbf{z}^j)\|^2, \quad \text{where} \quad \psi^*(\mathbf{z}^j) = \sum_{|\boldsymbol{\alpha}|=0}^{p} \mathbf{c}_{\boldsymbol{\alpha}}^* \boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z}^j). \tag{3.13}$$

If $\delta^* = 0$, the polynomial chaos expansion is a good approximation to the original deep neural network on the training dataset. However, the approximation loss of the testing dataset may be large because of the overfitting phenomena.

The objective function in (3.12) is an empirical approximation to the expected error

$$\mathbb{E}_{(\mathbf{z},\mathbf{y})}[\|\mathbf{y} - \psi(\mathbf{z})\|^2], \text{ where } \psi(\mathbf{z}) = \sum_{|\boldsymbol{\alpha}|=0}^{p} \mathbf{c}_{\boldsymbol{\alpha}} \boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z}). \tag{3.14}$$

According to Hoeffding's inequality [29], the expected error (3.14) is close to the empirical error (3.12) with a high probability. Consequently, the loss for ASNet with polynomial chaos expansion layer is bounded as follows.

Lemma 3.2. *Suppose that the optimal solution for solving problem* (3.12) *is* $\mathbf{c}_\alpha^*$*, the optimal polynomial chaos expansion is* $\psi^*(\mathbf{z})$*, and the optimal residue is* δ^**. Assume that there exist constants* a, b *such that for all* j*,* $\|\mathbf{y}^j - \psi^*(\mathbf{z}^j)\|^2 \in [a, b]$*. Then the loss of ASNet will be upper bounded*

$$\mathbb{E}_{\mathbf{z}}[(loss(\psi^*(\mathbf{z})))^2] \leq 2\mathbb{E}_{\mathbf{x}_0}[(c_0(\mathbf{x}_0))^2] + 2n_L(\delta^* + t) \quad w.p. \quad 1 - \gamma^*, \tag{3.15}$$

where t *is a user-defined threshold and* $\gamma^* = \exp(-\frac{2t^2 m_{PCE}}{(b-a)^2})$*.*

Proof. Since the cross entropy loss function is $\sqrt{n_L}$-Lipschitz continuous, we have

$$\mathbb{E}_{(\mathbf{y},\mathbf{z})}[(\text{loss}(\mathbf{y}) - \text{loss}(\psi^*(\mathbf{z})))^2] \leq n_L \mathbb{E}_{(\mathbf{y},\mathbf{z})}[\|\mathbf{y} - \psi^*(\mathbf{z})\|^2]. \tag{3.16}$$

Denote $\mathcal{T}^j = \|\mathbf{y}^j - \psi^*(\mathbf{z}^j)\|^2$ for $i = 1, \ldots, n_L$. $\{\mathcal{T}^j\}$ are independent under the assumption that the data samples are independent. By Hoeffding's inequality, for any constant t, it holds that

$$\mathbb{E}[\mathcal{T}] \leq \frac{1}{m_{\text{PCE}}} \sum_{j=1}^{m_{\text{PCE}}} \mathcal{T}^j + t \quad \text{w.p. } 1 - \gamma^* \tag{3.17}$$

with $\gamma^* = \exp(-\frac{2t^2 m_{\text{PCE}}}{(b-a)^2})$. Equivalently,

$$\mathbb{E}_{(\mathbf{y},\mathbf{z})}[\|\mathbf{y} - \psi^*(\mathbf{z})\|^2] \leq \delta^* + t \quad \text{w.p. } 1 - \gamma^*. \tag{3.18}$$

Consequently, there is

$$\begin{aligned} &\mathbb{E}_{\mathbf{z}}[(\text{loss}(\psi^*(\mathbf{z})))^2] \\ &\quad \leq 2\mathbb{E}_{\mathbf{y}}[(\text{loss}(\mathbf{y}))^2] + 2\mathbb{E}_{(\mathbf{y},\mathbf{z})}[(\text{loss}(\psi^*(\mathbf{z})) - \text{loss}(\mathbf{y}))^2] \\ &\quad \leq 2\mathbb{E}_{\mathbf{x}_0}[(c_0(\mathbf{x}_0))^2] + 2n_L(\delta^* + t) \quad \text{w.p. } 1 - \gamma^*. \end{aligned}$$

The last inequality follows from $c_0(\mathbf{x}_0) = c_l(\mathbf{x}_l) = \text{loss}(\mathbf{y})$, (3.16), and (3.18). This completes the proof. ■

Lemma 3.2 shows with a high probability $1-\gamma^*$, the expected error of ASNet without fine-tuning is bounded by the pretrained error of the original network, the accuracy loss in solving the polynomial chaos subproblem (3.13), and the number of classes n_L. The probability γ^* is controlled by the threshold t as well as the number of training samples m_{PCE}.

In practice, we always retrain ASNet for several epochs, and the accuracy of ASNet is beyond the scope of Lemma 3.2.

3.6. Structured retraining of ASNet.

The ASNet can be further compressed by various techniques such as network pruning and sharing [25], low-rank factorization [43, 36, 18], or data quantization [15, 12]. Denote $\boldsymbol{\theta}$ as the weights in ASNet and $\{\mathbf{x}_0^1, \ldots, \mathbf{x}_0^m\}$ as the training dataset. Here, $\boldsymbol{\theta}$ denotes all the parameters in the pre-model, active subspace layer, and the polynomial chaos expansion layer. We retrain the network by solving the following regularized optimization problem:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} \text{loss}(f(\boldsymbol{\theta}; \mathbf{x}_0^i)) + \lambda R(\boldsymbol{\theta}). \tag{3.19}$$

Here $(\mathbf{x}_0^i, \mathbf{y}^i)$ is a training sample, m is the total number of training samples, loss$(\cdot)$ is the cross entropy loss function, $R(\boldsymbol{\theta})$ is a regularization function, and λ is a regularization parameter. Different regularization functions can result in different model structures. For instance, an ℓ_1 regularizer $R(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$ [2, 50, 57] will return a sparse weight, an $\ell_{1,2}$-norm regularizer will result in a columnwise sparse weights, and a nuclear norm regularizer will result in low-rank weights. At each iteration, we solve (3.19) by a stochastic proximal gradient descent algorithm [53]:

$$\boldsymbol{\theta}^{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \quad (\boldsymbol{\theta} - \boldsymbol{\theta}^k)^T \mathbf{g}^k + \frac{1}{2\alpha_k}\|\boldsymbol{\theta} - \boldsymbol{\theta}^k\|_2^2 + \lambda R(\boldsymbol{\theta}). \tag{3.20}$$

Here $\mathbf{g}^k = \frac{1}{|\mathcal{B}_k|}\sum_{i\in\mathcal{B}_k} \nabla_{\boldsymbol{\theta}}\text{loss}(f(\boldsymbol{\theta};\mathbf{x}_0^i), \mathbf{y}^i)$ is the stochastic gradient, $\mathcal{B}_k$ is a batch at the kth step, and α_k is the stepsize.

In this work, we chose the ℓ_1 regularization $R(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$ to get sparse weights. In this case, problem (3.20) has a closed-form solution:

$$\boldsymbol{\theta}^{k+1} = \mathcal{S}_{\alpha_k\lambda}(\boldsymbol{\theta}^k - \alpha_k \mathbf{g}^k), \tag{3.21}$$

where $\mathcal{S}_\lambda(\mathbf{x}) = \mathbf{x} \odot \max(0, 1 - \lambda/|\mathbf{x}|)$ is a soft-thresholding operator.

4. Active subspace for universal adversarial attacks.

This section investigates how to generate a universal adversarial attack by the active subspace method. Given a function $f(\mathbf{x})$, the maximal perturbation direction is defined by

$$\mathbf{v}_\delta^* = \underset{\|\mathbf{v}\|_2\le\delta}{\operatorname{argmax}} \quad \mathbb{E}_{\mathbf{x}}[(f(\mathbf{x}+\mathbf{v}) - f(\mathbf{x}))^2]. \tag{4.1}$$

Here, δ is a user-defined perturbation upper bound. By the first order Taylor expansion, we have $f(\mathbf{x}+\mathbf{v}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T\mathbf{v}$, and problem (4.1) can be reduced to

$$\mathbf{v}_{AS} = \underset{\|\mathbf{v}\|_2=1}{\operatorname{argmax}} \quad \mathbb{E}_{\mathbf{x}}[(\nabla f(\mathbf{x})^T\mathbf{v})^2] = \underset{\|\mathbf{v}\|_2=1}{\operatorname{argmax}} \quad \mathbf{v}^T\mathbb{E}_{\mathbf{x}}[\nabla f(\mathbf{x})\nabla f(\mathbf{x})^T]\mathbf{v}. \tag{4.2}$$

The vector $\mathbf{v}_{AS}$ is exactly the dominant eigenvector of the covariance matrix of $\nabla f(\mathbf{x})$. The solution of (4.1) can be approximated by $+\delta\mathbf{v}_{AS}$ or $-\delta\mathbf{v}_{AS}$. Here, both $\mathbf{v}_{AS}$ and $-\mathbf{v}_{AS}$ are solutions of (4.2), but their effect on (4.1) is different.

Example 4.1. Consider a two-dimensional function $f(\mathbf{x}) = \mathbf{a}^T\mathbf{x} - b$ with $\mathbf{a} = [1, -1]^T$ and $b = 1$, and $\mathbf{x}$ follows a uniform distribution in a two-dimensional square domain $[0, 1]^2$, as shown in Figure 4(a). It follows from direct computations that $\nabla f(\mathbf{x}) = \mathbf{a}$ and the covariance matrix $\mathbf{C} = \mathbf{a}\mathbf{a}^T$. The dominant eigenvector of $\mathbf{C}$ or the active subspace direction is $\mathbf{v}_{AS} = \mathbf{a}/\|\mathbf{a}\|_2 = [1/\sqrt{2}, -1/\sqrt{2}]$. We apply $\mathbf{v}_{AS}$ to perturb $f(\mathbf{x})$ and plot $f(\mathbf{x} + \delta\mathbf{v}_{AS})$ in Figure 4(b), which shows a significant difference even for a small permutation $\delta = 0.3$. Furthermore, we plot the perturbed function along the first principal component direction $\mathbf{w}_1 = [1/\sqrt{2}, 1/\sqrt{2}]^T$ in Figure 4(c). Here, $\mathbf{w}_1$ is the eigenvector of the covariance matrix $\mathbb{E}_{\mathbf{x}}[\mathbf{x}\mathbf{x}^T] = \begin{bmatrix} 1/3 & 1/4 \\ 1/4 & 1/3 \end{bmatrix}$. However, $\mathbf{w}_1$ does not result in any perturbation because $\mathbf{a}^T\mathbf{w}_1 = 0$. This example indicates the difference between the active subspace and principal component analysis: the active subspace direction can capture the sensitivity information of $f(\mathbf{x})$, whereas the principal component is independent of $f(\mathbf{x})$.

4.1. Universal perturbation of deep neural networks.

Given a dataset $\mathcal{D}$ and a classification function $j(\mathbf{x})$ that maps an input sample to an output label, the universal perturbation

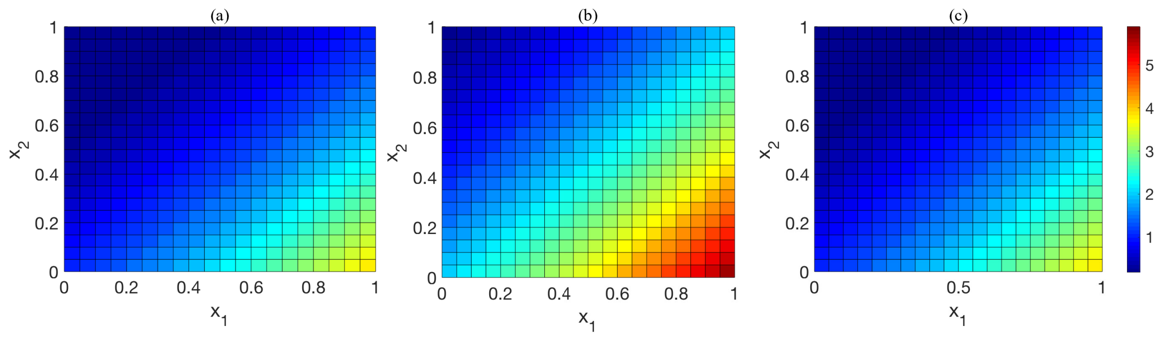

Figure 4. *Perturbations along the directions of an active subspace direction and of principal component, respectively.* (a) *The function* $f(\mathbf{x}) = \mathbf{a}^T\mathbf{x} - b$. (b) *The perturbed function along the active subspace direction.* (c) *The perturbed function along the principal component analysis direction.*

seeks for a vector $\mathbf{v}^*$ whose norm is upper bounded by δ such that the class label can be perturbed with a high probability, i.e.,

$$\mathbf{v}^* = \underset{\|\mathbf{v}\|\le\delta}{\operatorname{argmax}}\ \mathrm{prob}_{\mathbf{x}\in\mathcal{D}}[j(\mathbf{x}+\mathbf{v}) \neq j(\mathbf{x})] = \underset{\|\mathbf{v}\|\le\delta}{\operatorname{argmax}}\ \mathbb{E}_{\mathbf{x}}[1_{j(\mathbf{x}+\mathbf{v})\neq j(\mathbf{x})}], \tag{4.3}$$

where 1_d equals one if the condition d is satisfied and zero otherwise. Solving problem (4.3) directly is challenging because both 1_d and $j(\mathbf{x})$ are discontinuous. By replacing $j(\mathbf{x})$ with the loss function $c(\mathbf{x}) = \mathrm{loss}(f(\mathbf{x}))$ and the indicator function 1_d with a quadratic function, we reformulate problem (4.3) as

$$\max_{\mathbf{v}}\quad \mathbb{E}_{\mathbf{x}}[(c(\mathbf{x}+\mathbf{v}) - c(\mathbf{x}))^2] \quad \text{s.t.} \quad \|\mathbf{v}\|_2 \le \delta. \tag{4.4}$$

The ball-constrained optimization problem (4.4) can be solved by various numerical techniques such as the spectral gradient descent method [6] and the limited-memory projected quasi-Newton [51]. However, these methods can only guarantee convergence to a local stationary point. Instead, we are interested in computing a direction that can achieve a better objective value by a heuristic algorithm.

4.2. Recursive projection method.

Using the first order Taylor expansion $c(\mathbf{x}+\mathbf{v}) \approx c(\mathbf{x}) + \mathbf{v}^T\nabla c(\mathbf{x})$, we reformulate problem (4.4) as a ball-constrained quadratic problem:

$$\max_{\mathbf{v}}\quad \mathbf{v}^T\mathbb{E}_{\mathbf{x}}[\nabla c(\mathbf{x})\nabla c(\mathbf{x})^T]\mathbf{v} \quad \text{s.t.} \quad \|\mathbf{v}\|_2 \le \delta. \tag{4.5}$$

Problem (4.5) is easy to solve because its closed-form solution is exactly the dominant eigenvector of the covariance matrix $\mathbf{C} = \mathbb{E}_{\mathbf{x}}[\nabla c(\mathbf{x})\nabla c(\mathbf{x})^T]$ or the first active subspace direction. However, the dominant eigenvector in (4.5) may not be efficient because $c(\mathbf{x})$ is nonlinear. Therefore, we compute $\mathbf{v}$ recursively by

$$\mathbf{v}^{k+1} = \mathrm{proj}(\mathbf{v}^k + s^k d_{\mathbf{v}}^k), \tag{4.6}$$

where $\mathrm{proj}(\mathbf{v}) = \mathbf{v} \times \min(1, \delta/\|\mathbf{v}\|_2)$, s^k is the stepsize, and $d_{\mathbf{v}}^k$ is approximated by

$$d_{\mathbf{v}}^k = \underset{d_{\mathbf{v}}}{\operatorname{argmax}}\quad d_{\mathbf{v}}^T\mathbb{E}_{\mathbf{x}}\left[\nabla c\left(\mathbf{x}+\mathbf{v}^k\right)\nabla c\left(\mathbf{x}+\mathbf{v}^k\right)^T\right] d_{\mathbf{v}} \text{ s.t. } \|d_{\mathbf{v}}\|_2 \le 1. \tag{4.7}$$

Namely, $d_{\mathbf{v}}^k$ is the dominant eigenvector of $\mathbf{C}^k = \mathbb{E}_{\mathbf{x}}[\nabla c\left(\mathbf{x}+\mathbf{v}^k\right)\nabla c\left(\mathbf{x}+\mathbf{v}^k\right)^T]$. Because $d_{\mathbf{v}}^k$ maximizes the changes in $\mathbb{E}_{\mathbf{x}}[(c(\mathbf{x}+\mathbf{v}+d_{\mathbf{v}}) - c(\mathbf{x}+\mathbf{v}))^2]$, we expect that the attack ratio keeps increasing, i.e., $r(\mathbf{v}^{k+1};\mathcal{D}) \geq r(\mathbf{v}^k;\mathcal{D})$, where

$$r(\mathbf{v};\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}^i\in\mathcal{D}} 1_{j(\mathbf{x}^i+\mathbf{v})\neq j(\mathbf{x}^i)}. \tag{4.8}$$

The backtracking line search approach [3] is employed to choose s^k such that the attack ratio of $\mathbf{v}^k + s^k d_{\mathbf{v}}^k$ is higher than the attack ratio of both $\mathbf{v}^k$ and $\mathbf{v}^k - s^k d_{\mathbf{v}}^k$, i.e.,

$$s^k = \min_i \left\{ s_{i,t}^k : r\left(\mathbf{v}_{i,t}^{k+1};\mathcal{D}\right) > \max\left(r\left(\mathbf{v}_{i,-t}^{k+1};\mathcal{D}\right), r\left(\mathbf{v}^k;\mathcal{D}\right)\right)\right\}, \tag{4.9}$$

where $s_{i,t}^k = (-1)^t s_0\gamma^i$, $t\in\{1,-1\}$, s_0 is the initial stepsize, $\gamma < 1$ is the decrease ratio, and $\mathbf{v}_{i,t}^{k+1} = \text{proj}(\mathbf{v}^k + s_{i,t}^{k+1} d_{\mathbf{v}}^k)$. If such a stepsize s^k exists, we update $\mathbf{v}^{k+1}$ by (4.6) and repeat the process. Otherwise, we record the number of failures and stop the algorithm when the number of failure is greater than a threshold.

The overall flow is summarized in Algorithm 4.1. In practice, instead of using the whole dataset to train this attack vector, we use a subset $\mathcal{D}^0$. The impact for different numbers of samples is discussed in section 5.2.2. Further, at line 3 in Algorithm 4.1, we only use the data samples that can not be attacked by $\mathbf{v}^k$ to train the next attack vector $\mathbf{v}^{k+1}$.

Algorithm 4.1. Recursive active subspace for universal attacks.

Input: A pretrained deep neural network denoted as $c(\mathbf{x})$, a classification oracle $j(\mathbf{x})$, a training dataset $\mathcal{D}^0$, an upper bound for the attack vector δ, an initial stepsize s_0, a decrease ratio $\gamma < 1$, and the parameter in the stopping criterion α.

```
1: Initialize the attack vector as $\mathbf{v}^0 = 0$.
2: for $k = 0, 1, \ldots$ do
3:   Select the training dataset as $\mathcal{D} = \{\mathbf{x}^i + \mathbf{v}^k : \mathbf{x}^i \in \mathcal{D}^0$ and $j(\mathbf{x}^i+\mathbf{v}^k) = j(\mathbf{x}^i)\}$; then
     compute the dominate active subspace direction $d\mathbf{v}$ by Algorithm 3.2.
4:   for $i = 0, 1, \ldots I$ do
5:     Let $s_{i,\pm}^k = (-1)^{\pm} s_0\gamma^i$ and $\mathbf{v}_{i,\pm}^{k+1} = \text{proj}(\mathbf{v}^k + s_{i,\pm}^{k+1} d_{\mathbf{v}}^k)$ . Compute the attack ratios
       $r(\mathbf{v}_{i,1}^{k+1})$ and $r(\mathbf{v}_{i,-1}^{k+1})$ by (4.8).
6:     If either $r(\mathbf{v}_{i,1}^{k+1})$ or $r(\mathbf{v}_{i,-1}^{k+1})$ is greater than $r(\mathbf{v}^k)$, stop the process. Return $s^k =$
       $(-1)^t s_{i,1}^k$, where $t = 1$ if $r(\mathbf{v}_{i,1}^{k+1}) \geq r(\mathbf{v}_{i,-1}^{k+1})$ and $t = -1$ otherwise.
7:   end for
     If no stepsize $s^k$ is returned, let $s^k = s_0 r^I$ and record this step as a failure. Compute
     the next iteration $\mathbf{v}^{k+1}$ by the projection (4.6).
8:   If the number of failure is greater the threshold $\alpha$, stop.
9: end for
   Output: The universal active adversarial attack vector $\mathbf{v}_{AS}$.
```

5. Numerical experiments.

In this section, we show the power of active subspace in revealing the number of active neurons, compressing neural networks, and computing the

Table 1

Comparison of number of neurons r of VGG-19 on CIFAR-10. For the storage speedup, higher is bettter. For the accuracy reduction before or after fine-tuning, lower is better.

	$r = 25$				$r = 50$				$r = 75$			
	ϵ	Storage	Accu. Before	Reduce After	ϵ	Storage	Accu. Before	Reduce After	ϵ	Storage	Accu. Before	Reduce After
ASNet(5)	0.34	**20.7**×	7.06	2.82	0.18	14.4×	4.40	1.82	0.11	11.0×	**3.64**	**1.66**
ASNet(6)	0.24	**12.8**×	2.14	0.59	0.11	10.1×	1.62	0.27	0.05	8.3×	**1.40**	**0.21**
ASNet(7)	0.15	**9.3**×	0.79	0.11	0.06	7.8×	**0.63**	**-0.10**	0.03	6.7×	0.77	0.00

universal adversarial perturbation. All codes are implemented in PyTorch and are available online.[1]

5.1. Structural analysis and compression. We test the ASNet constructed by Algorithm 3.1 and set the polynomial order as $p = 2$, the number of active neurons as $r = 50$, and the threshold in (3.4) as $\epsilon = 0.05$ on default. Inspired by the knowledge distillation [28], we retrain all the parameters in the ASNet by minimizing the following loss function:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{m} \beta H\left(\text{ASNet}_{\boldsymbol{\theta}}(\mathbf{x}_0^i), f(\mathbf{x}_0^i)\right) + (1-\beta) H\left(\text{ASNet}_{\boldsymbol{\theta}}(\mathbf{x}_0^i), \mathbf{y}^i\right).$$

Here, the cross entropy $H(\mathbf{p}, \mathbf{q}) = \sum_j s(\mathbf{p})_j \log s(\mathbf{q})_j$, the softmax function $s(\mathbf{x})_j = (\exp(x_j))/(\sum_j \exp(x_j))$, and the parameter $\beta = 0.1$ on default. We retrain ASNet for 50 epochs by ADAM [34]. The stepsizes for the pre-model are set as 10^{-4} and 10^{-3} for VGG-19 and ResNet, and the stepsize for the active subspace layer and the polynomial chaos expansion layer is set as 10^{-5}, respectively.

We also seek for sparser weights in ASNet by the proximal stochastic gradient descent method in section 3.6. On default, we set the stepsize as 10^{-4} for the pre-model and 10^{-5} for the active subspace layer and the polynomial chaos expansion layer. The maximal epoch is set as 100. The obtained sparse model is denoted ASNet-s.

In all figures and tables, the numbers in the bracket of ASNet($\cdot$) or ASNet-s($\cdot$) indicate the index of a cut-off layer. We report the performance for different cut-off layers in terms of *accuracy, storage, and computational complexities.*

5.1.1. Choices of parameters. We first show the influence of number of reduced neurons r, tolerance ϵ, and cutting-off layer index l of VGG-19 on CIFAR-10 in Table 1. The VGG-19 can achieve 93.28% testing accuracy with 76.45 MB storage consumption. Here, $\epsilon = \frac{\lambda_{r+1}+\cdots+\lambda_n}{\lambda_1+\cdots+\lambda_n}$. For different choices of r, we display the corresponding tolerance ϵ, the storage speedup compared with the original teacher network, and the testing accuracy reduction for ASNet before and after fine-tuning compared with the original teacher network.

Table 1 shows that when the cutting-off layer is fixed, a larger r usually results in a smaller tolerance ϵ and a smaller accuracy reduction but also a smaller storage speedup. This is corresponding to Lemma 3.1 that the error of ASNet before fine-tuning is upper bounded by $O(\epsilon)$. Comparing $r = 50$ with $r = 75$, we find that $r = 50$ can achieve almost the same

[1]https://github.com/chunfengc/ASNet.

accuracy with $r = 75$ with a higher storage speedup. $r = 50$ can even achieve better accuracy than $r = 75$ in layer 7 probably because of overfitting. This guides us to chose $r = 50$ in the following numerical experiments. For different layers, we see a later cutting-off layer index can produce a lower accuracy reduction but a smaller storage speedup. In other words, the choice of layer index is a trade-off between accuracy reduction and storage speedup.

5.1.2. Efficiency of active subspace. We show the effectiveness of ASNet constructed by steps 1–3 of Algorithm 3.1 without fine-tuning. We investigate the following three properties.

(1) *Redundancy of neurons.* The distributions of the first 200 singular values of the matrix $\hat{\mathbf{G}}$ (defined in (3.7)) are plotted in Figure 5(a). The singular values decrease almost exponentially for layers $l \in \{4, 5, 6, 7\}$. Although the total numbers of neurons are 8192, 16384, 16384, and 16384, the numbers of active neurons are only 105, 84, 54, and 36, respectively.

(2) *Redundancy of the layers.* We cut off the deep neural network at an intermediate layer and replace the subsequent layers with one simple logistic regression [30]. As shown by the red bar in Figure 5(b), the logistic regression can achieve relatively high accuracy. This verifies that the features trained from the first few layers already have a high expression power since replacing all subsequent layers with a simple expression loses little accuracy.

(3) *Efficiency of the active subspace and polynomial chaos expansion.* We compare the proposed active subspace layer with the principal component analysis [31] in projecting the high-dimensional neuron to a low-dimensional space, and also compare the polynomial chaos expansion layer with logistic regression in terms of their efficiency to extract class labels from the low-dimensional variables. Figure 5(b) shows that the combination of active subspace and polynomial chaos expansion can achieve the best accuracy.

5.1.3. CIFAR-10. We continue to present the results of ASNet and ASNet-s on CIFAR-10 by two widely used networks: VGG-19 and ResNet-110 in Tables 2 and 3, respectively. The second column shows the testing accuracy for the corresponding network. We report the storage and computational costs for the pre-model, post-model (i.e., active subspace plus polynomial chaos expansion for ASNet and ASNet-s), and overall results, respectively. For both examples, ASNet and ASNet-s can achieve a similar accuracy with the teacher network

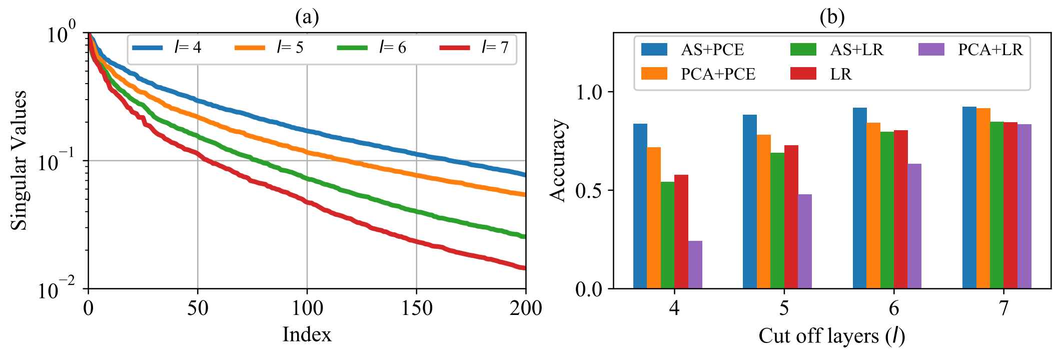

Figure 5. *Structural analysis of VGG-19 on the CIFAR-10 dataset.* (a) *The first 200 singular values for layers $4 \leq l \leq 7$.* (b) *The accuracy (without any fine-tuning) obtained by active subspace (AS) and polynomial chaos expansions (PCE) compared with principal component analysis (PCA) and logistic regression (LR).*

Table 2

*Accuracy and storage on VGG-*19 *for CIFAR-*10*. Here, "Pre-M" denotes the pre-model, i.e., layers* 1 *to* l *of the original deep neural networks, and "AS" and "PCE" denote the active subspace and polynomial chaos expansion layer, respectively.*

Network	Accuracy	Storage (MB)			Flops (10^6)		
VGG-19	93.28%	76.45			398.14		
		Pre-M	AS+PCE	Overall	Pre-M	AS+PCE	Overall
ASNet(5)	91.46%	2.12	3.18 (23.41×)	5.30 (14.43×)	115.02	0.83 (340.11×)	115.85 (3.44×)
ASNet-s(5)	90.40%	1.14 (1.86×)	2.05 (36.33×)	**3**.19 **(**23.98×)	54.03 (2.13×)	0.54 (527.91×)	**5**4.56 **(**7.30×)
ASNet(6)	93.01%	4.38	3.18 (22.70×)	7.55 (10.12×)	152.76	0.83 (294.76×)	153.60 (2.59×)
ASNet-s(6)	91.08%	1.96 (2.24×)	1.81 (39.73×)	3.77 (20.27×)	67.37 (2.27×)	0.48 (515.98×)	67.85 (5.87×)
ASNet(7)	**9**3.38%	6.63	3.18 (21.99×)	9.80 (7.80×)	190.51	0.83 (249.41×)	191.35 (2.08×)
ASNet-s(7)	90.87%	2.61 (2.54×)	1.91 (36.64×)	4.52 (16.92×)	80.23 (2.37×)	0.50 (415.68×)	80.73 (4.93×)

Table 3

*Accuracy and storage on ResNet-*110 *for CIFAR-*10*. Here, "Pre-M" denotes the pre-model, i.e., layers* 1 *to* l *of the original deep neural networks, and "AS" and "PCE" denote the active subspace and polynomial chaos expansion layer, respectively.*

Network	Accuracy	Storage (MB)			Flops (10^6)		
ResNet-110	93.78%	6.59			252.89		
		Pre-M	AS+PCE	Overall	Pre-M	AS+PCE	Overall
ASNet(61)	89.56%	1.15	1.61 (3.37×)	2.77 (2.38×)	140.82	0.42 (265.03×)	141.24 (1.79×)
ASNet-s(61)	89.26%	0.83 (1.39×)	1.23 (4.41×)	**2**.06 **(**3.19×)	104.05 (1.35×)	0.32 (346.82×)	**1**04.37 **(**2.42×)
ASNet(67)	90.16%	1.37	1.61 (3.24×)	2.98 (2.21×)	154.98	0.42 (231.55×)	155.40 (1.63×)
ASNet-s(67)	89.69%	1.00 (1.36×)	1.22 (4.29×)	2.22 (2.97×)	116.38 (1.33×)	0.32 (306.72×)	116.70 (2.17×)
ASNet(73)	**9**0.48%	1.58	1.61 (3.11×)	3.19 (2.06×)	169.13	0.42 (198.07×)	169.55 (1.49×)
ASNet-s(73)	90.02%	1.18 (1.34×)	1.16 (4.32×)	2.34 (2.82×)	128.65 (1.31×)	0.30 (275.74×)	128.96 (1.96×)

yet with much smaller storage and computational cost. For VGG-19, ASNet achieves 14.43× storage savings and 3.44× computational reduction; ASNet-s achieves 23.98× storage savings and 7.30× computational reduction. For most ASNet and ASNet-s networks, the storage and computational costs of the post-models achieve significant performance boosts by our proposed network structure changes. It is not surprising to see that increasing the layer index (i.e., cutting off the deep neural network at a later layer) can produce a higher accuracy. However, increasing the layer index also results in a smaller compression ratio. In other words, the choice of layer index is a trade-off between the accuracy reduction and the compression ratio.

For ResNet-110, our results are not as good as those on VGG-19. We find that the eigenvalues for its covariance matrix are not exponentially decreasing as those of VGG-19, which results in a large number of active neurons or a large error ϵ when fixing $r = 50$. A possible reason is that ResNet updates as $\mathbf{x}_{l+1} = \mathbf{x}_l + f_l(\mathbf{x}_l)$. Hence, the partial gradient $\partial \mathbf{x}_{l+1}/\partial \mathbf{x}_l = I + \nabla f_l(\mathbf{x}_l)$ is less likely to be low-rank.

5.1.4. CIFAR-100. Next, we present the results of VGG-19 and ResNet-110 on CIFAR-100 in Tables 4 and 5, respectively. On VGG-19, ASNet can achieve 7.45× storage savings and 2.08× computational reduction, and ASNet-s can achieve 9.06× storage savings and 2.73× computational reduction. The accuracy loss is negligible for VGG-19 but larger for ResNet-110. The performance boost of ASNet is obtained by just changing the network structures and without any model compression (e.g., pruning, quantization, or low-rank factorization).

5.2. Universal adversarial attacks. This subsection demonstrates the effectiveness of active subspace in identifying a universal adversarial attack vector. We denote the result generated by Algorithm 4.1 as "AS" and compare it with the "UAP" method in [40] and with "random" Gaussian distribution vector. The parameters in Algorithm 4.1 are set as $\alpha = 10$ and $\delta = 5, \ldots, 10$. The default parameters of UAP are applied except for the maximal iteration. In the implementation of [40], the maximal iteration is set as infinity, which is time-consuming when the training dataset or the number of classes is large. In our experiments, we set the maximal iteration as 10. In all figures and tables, we report the average attack ratio and CPU time in training out of ten repeated experiments with different training datasets. A higher attack ratio means the corresponding algorithm is better at fooling the given deep neural network. The datasets are chosen in two ways. We first test data points from one class (e.g., trousers in Fashion-MNIST) because these data points share lots of common features and have a higher probability to be attacked by a universal perturbation vector. We then conduct experiments on the whole dataset to show our proposed algorithm can also provide better performance compared with the baseline even if the dataset has diverse features.

Table 4

Accuracy and storage on VGG-19 for CIFAR-100. Here, "Pre-M" denotes the pre-model, i.e., layers 1 to l of the original deep neural networks, and "AS" and "PCE" denote the active subspace and polynomial chaos expansion layer, respectively.

Network	Top-1	Top-5	Storage (MB)			Flops (10^6)		
VGG-19	71.90%	89.57%	76.62			398.18		
			Pre-M	AS+PCE	Overall	Pre-M	AS+PCE	Overall
ASNet(7)	70.77%	91.05%	6.63	3.63 (19.23×)	10.26 (7.45×)	190.51	0.83 (249.41×)	191.35 (2.08×)
ASNet-s(7)	70.20%	90.90%	5.20 (1.27×)	3.24 (21.56×)	8.44 (9.06×)	144.81 (1.32×)	0.85 (244.57×)	**145.66 (2.73×)**
ASNet(8)	69.50%	90.15%	8.88	1.29 (52.50×)	10.17 (7.52×)	228.26	0.22 (779.04×)	228.48 (1.74×)
ASNet-s(8)	69.17%	89.73%	6.87 (1.29×)	1.22 (55.36×)	**8.09 (9.45×)**	172.69 (1.32×)	0.32 (530.92×)	173.01 (2.30×)
ASNet(9)	**72.00%**	**90.61%**	13.39	2.07 (30.49×)	15.46 (4.95×)	247.14	0.42 (357.10×)	247.56 (1.61×)
ASNet-s(9)	71.38%	90.28%	9.38 (1.43×)	1.94 (32.49×)	11.32 (6.75×)	183.27 (1.35×)	0.51 (296.74×)	183.78 (2.17×)

Table 5

*Accuracy and storage on ResNet-*110 *for CIFAR-*100*. Here, "Pre-M" denotes the pre-model, i.e., layers* 1 *to l of the original deep neural networks, and "AS" and "PCE" denote the active subspace and polynomial chaos expansion layer, respectively.*

Network	Top-1	Top-5	Storage (MB)			Flops (10^6)		
ResNet-110	71.94%	91.71 %	6.61			252.89		
			Pre-M	AS+PCE	Overall	Pre-M	AS+PCE	Overall
ASNet(75)	63.01%	88.55%	1.79	1.29 (3.73×)	3.08 (2.14×)	172.67	0.22 (367.88×)	172.89 (1.46×)
ASNet-s(75)	63.16%	88.65%	1.47 (1.22×)	1.20 (3.99×)	**2**.67 **(**2.46×)	143.11 (1.21×)	0.31 (254.69×)	**1**43.42 **(**1.76×)
ASNet(81)	65.82%	90.02%	2.64	1.29 (3.07×)	3.93 (1.68×)	186.83	0.22 (302.96×)	187.04 (1.35×)
ASNet-s(81)	65.73%	89.95%	2.20 (1.20×)	1.21 (3.27×)	3.41 (1.93×)	155.61 (1.20×)	0.32 (208.38×)	155.93 (1.62×)
ASNet(87)	**6**7.71%	**9**0.17%	3.48	1.29 (2.41×)	4.77 (1.38×)	200.98	0.22 (238.04×)	201.20 (1.26×)
ASNet-s(87)	67.65%	90.10%	2.91 (1.20×)	1.21 (2.56×)	4.12 (1.60×)	166.50 (1.21×)	0.32 (163.50×)	166.81 (1.52×)

5.2.1. Fashion-MNIST. Firstly, we present the adversarial attack result on Fashion-MNIST by a 4-layer neural network. There are two convolutional layers with kernel size equal to 5×5. The size of output channels for each convolutional layer is 20 and 50, respectively. Each convolutional layer is followed by a ReLU activation layer and a max pooling layer with a kernel size of 2×2. There are two fully connected layers. The first fully connected layer has an input feature 800 and an output feature 500.

Figure 6 presents the attack ratio of our active subspace method compared with the baseline UAP method [40] and Gaussian random vectors. The top figures show the results for just one class (i.e., trouser), and the bottom figures show the results for all ten classes. For all perturbation norms, the active subspace method can achieve around 30% higher attack ratio than UAP more than 10 times faster. This verifies that the active subspace method has better universal representation ability compared with UAP because the active subspace can find a universal direction while UAP solves data-dependent subproblems independently. By the active subspace approach, the attack ratio for the first class and the whole dataset are around 100% and 75%, respectively. This coincides with our intuition that the data points in one class have higher similarity than data points from different classes.

In Figure 7, we plot one image from Fashion-MNIST and its perturbation by the active-subspace attack vector. The attacked image in Figure 7(c) still looks like a trouser for a human. However, the deep neural network misclassifies it as a t-shirt/top.

5.2.2. CIFAR-10. Next, we show the numerical results of attacking VGG-19 on CIFAR-10. Figure 8 compares the active subspace method with the baseline UAP and Gaussian random vectors. The top figures show the results by the dataset in the first class (i.e., automobile), and the bottom figures show the results for all ten classes. For both cases, the proposed active subspace attack can achieve 20% higher attack ratios three times faster than UAP. This is similar to the results in Fashion-MNIST because the active subspace has a better ability to capture the global information.

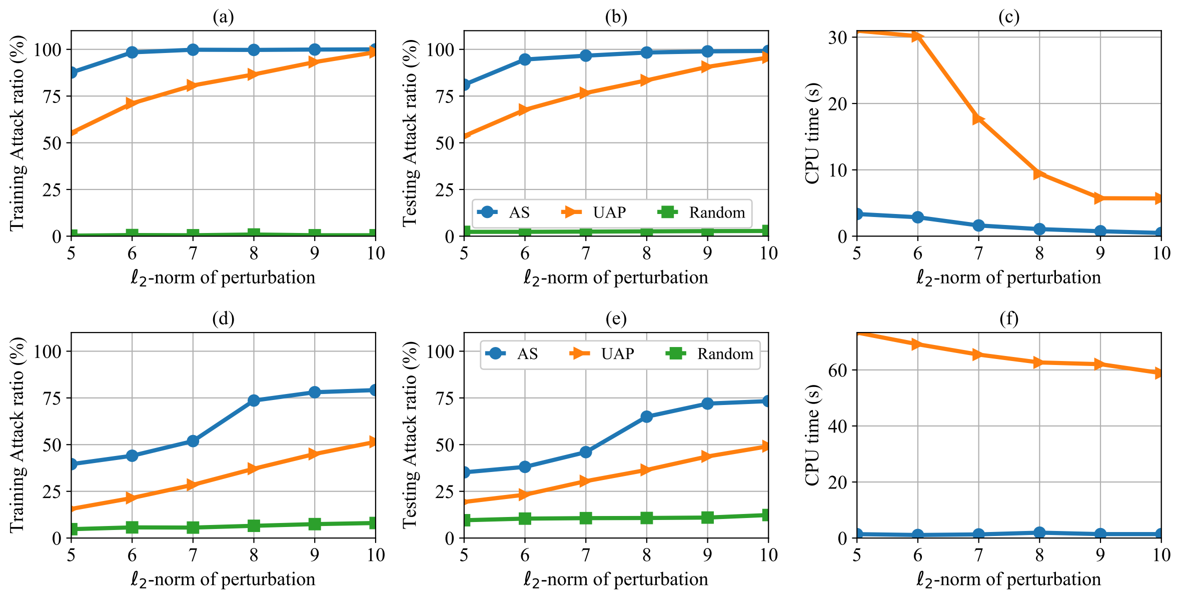

Figure 6. *Universal adversarial attacks for the Fashion-MINST with respect to different ℓ_2-norms.* (a)–(c): *The results for attacking one class dataset.* (d)–(f): *The results for attacking the whole dataset.*

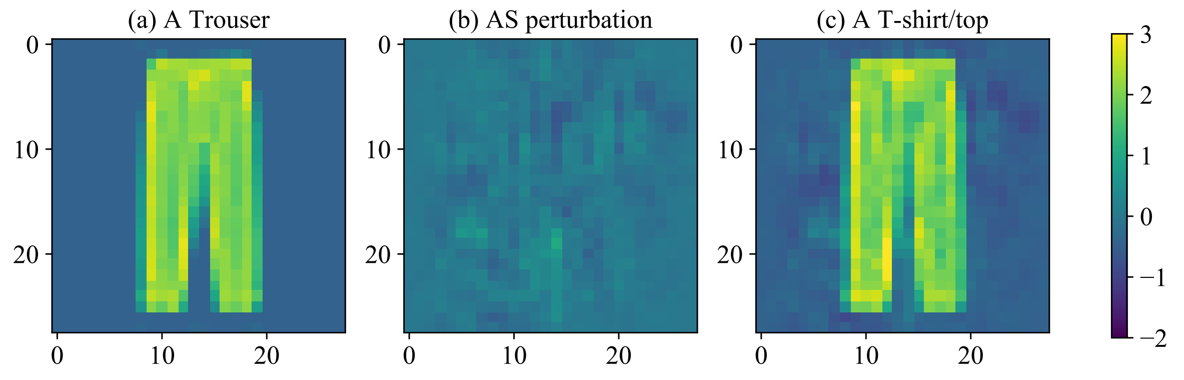

Figure 7. *The effect of our attack method on one data sample in the Fashion-MNIST dataset.* (a) *A trouser from the original dataset.* (b) *An active subspace perturbation vector with the ℓ_2-norm equal to* 5. (c) *The perturbed sample is misclassified as a t-shirt/top by the deep neural network.*

We further show the effects of *different numbers of training samples* in Figure 9. When the number of samples is increased, the testing attack ratio is getting better. In our numerical experiments, we set the number of samples as 100 for one-class experiments and 200 for all-classes experiments.

We continue to show the *cross-model* performance on four different ResNet networks and one VGG network. We test the performance of the attack vector trained from one model on all other models. Each row in Table 6 shows the results on the same deep neural network, and each column shows the results of the same attack vector. It shows that ResNet-20 is easier

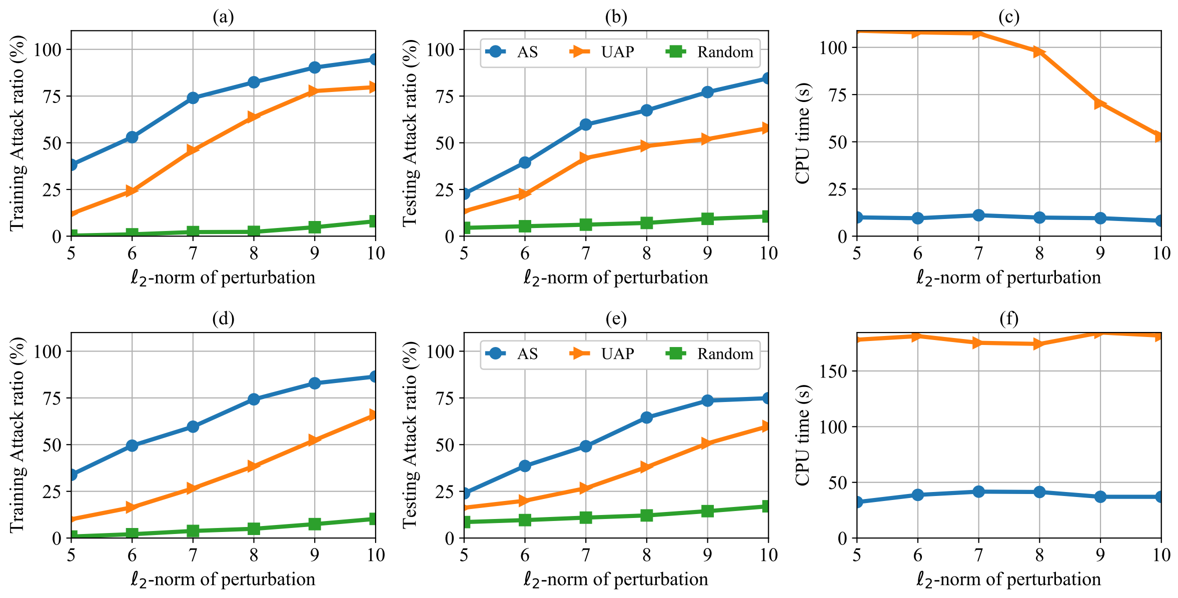

Figure 8. *Universal adversarial attacks of VGG-*19 *on CIFAR-*10 *with respect to different ℓ_2-norm perturbations.* (a)–(c): *The training attack ratio, the testing attack ratio, and the CPU time in seconds for attacking one class dataset.* (d)–(f): *The results for attacking ten classes' dataset together.*

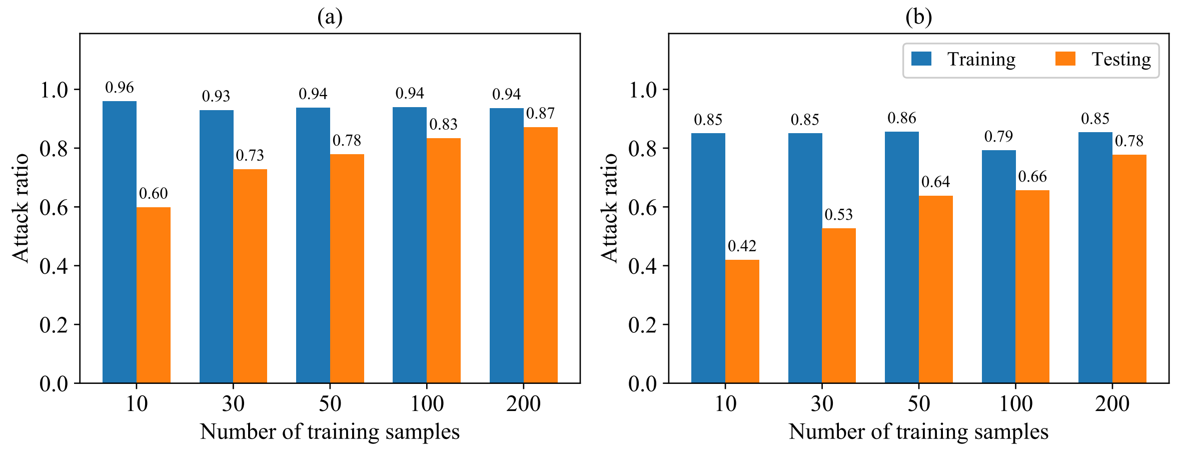

Figure 9. *Adversarial attack of VGG-*19 *on CIFAR-*10 *with different number of training samples. The ℓ_2-norm perturbation is fixed as* 10. (a) *The results of attacking the dataset from the first class.* (b) *The results of attacking the whole dataset with* 10 *classes.*

to be attacked compared with other models. This agrees with our intuition that a simple network structure such as ResNet-20 is less robust. On the contrary, VGG-19 is the most robust. The success of cross-model attacks indicates that these neural networks could find a similar feature.

Table 6
Cross-model performance for CIFAR-10.

	ResNet-20	ResNet-44	ResNet-56	ResNet-110	VGG-19
ResNet-20	**91.35%**	87.74%	86.28%	87.38%	81.16%
ResNet-44	84.75%	**92.28%**	87.03%	85.44%	83.44%
ResNet-56	83.63%	86.67%	**90.15%**	87.39%	84.38%
ResNet-110	71.02%	77.58%	74.19%	**92.77%**	77.32%
VGG-19	53.61%	59.74%	61.49%	66.29%	**80.02%**

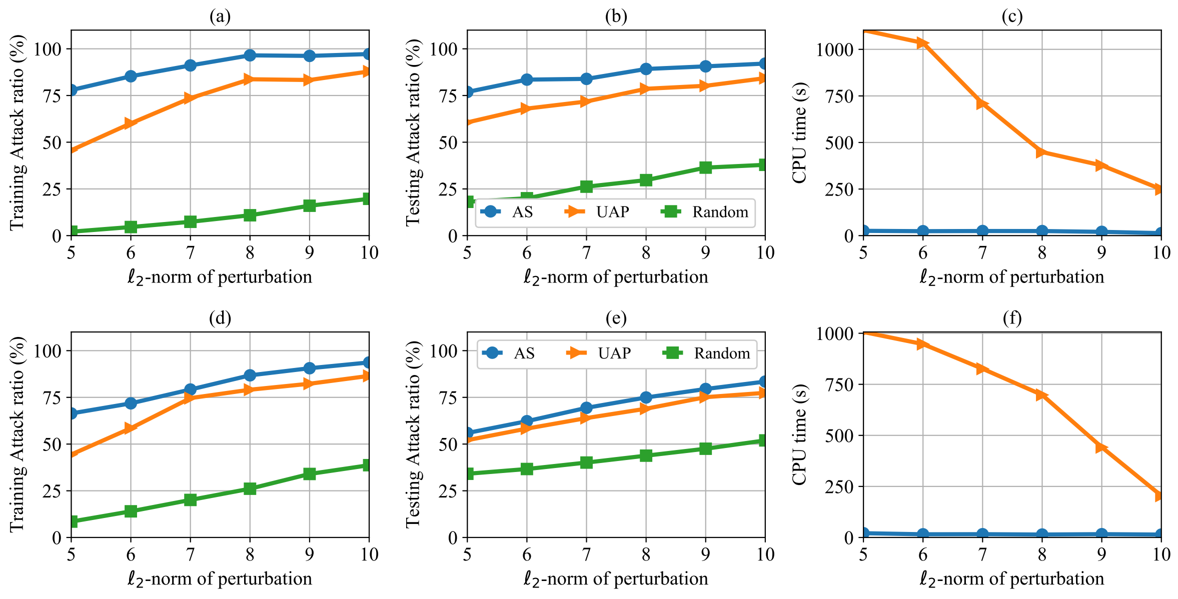

Figure 10. *Results for universal adversarial attack for CIFAR-*100 *with respect to different ℓ_2-norm perturbations.* (a)–(c): *The results for attacking the dataset from the first class.* (d)–(f): *The results for attacking ten classes' dataset together.*

5.2.3. CIFAR-100. Finally, we show the results on CIFAR-100 for both the first class (i.e., dolphin) and all classes. Similar to Fashion-MNIST and CIFAR-10, Figure 10 shows that active subspace can achieve higher attack ratios than both UAP and Gaussian random vectors. Further, compared with CIFAR-10, CIFAR-100 is easier to be attacked partially because it has more classes.

We summarize the results for different datasets in Table 7. The second column shows the number of classes in the dataset. In terms of testing attack ratio for the whole dataset, active subspace achieves 24.2%, 15%, and 6.1% higher attack ratios than UAP for Fashion-MNIST, CIFAR-10, and CIFAR-100, respectively. In terms of the CPU time, active subspace achieves 42×, 5×, and 14× speedup over UAP on the Fashion-MNIST, CIFAR-10, and CIFAR-100, respectively.

Table 7

Summary of the universal attack for different datasets by the active subspace compared with UAP and the random vector. The norm of perturbation is equal to 10.

		Training Attack ratio			Testing Attack ratio			CPU time (s)	
	# Class	AS	UAP	Rand	AS	UAP	Rand	AS	UAP
Fashion-MNIST	1	100.0%	93.6%	1.8%	**9**8.0%	91.3%	3.0%	**0**.15	5.49
	10	79.2%	51.5%	8.0%	**7**3.3%	49.1%	12.3%	**1**.40	58.85
CIFAR-10	1	94.7%	79.8%	8.0%	**8**4.5%	57.9%	10.6%	**8**.18	52.83
	10	86.5%	65.9%	10.2%	**7**4.9%	59.9%	17.0%	**37**.01	181.72
CIFAR-100	1	97.2%	87.9%	19.7%	**9**2.1%	84.3%	37.9%	**1**3.32	248.78
	100	93.7%	86.5%	38.7%	**8**3.5%	77.4%	52.0%	**1**4.32	204.50

6. Conclusions and discussions.

This paper has analyzed deep neural networks by the active subspace method originally developed for dimensionality reduction of uncertainty quantification. We have investigated two problems: how many neurons and layers are necessary (or important) in a deep neural network, and how can we generate a universal adversarial attack vector that can be applied to a set of testing data? Firstly, we have presented a definition of "the number of active neurons" and have shown its theoretical error bounds for model reduction. Our numerical study has shown that many neurons and layers are not needed. Based on this observation, we have proposed a new network called ASNet by cutting off the whole neural network at a proper layer and replacing all subsequent layers with an active subspace layer and a polynomial chaos expansion layer. The numerical experiments show that the proposed deep neural network structural analysis method can produce a new network with significant storage savings and computational speedup yet with little accuracy loss. Our methods can be combined with existing model compression techniques (e.g., pruning, quantization, and low-rank factorization) to develop compact deep neural network models that are more suitable for the deployment on resource-constrained platforms. Secondly, we have applied the active subspace to generate a universal attack vector that is independent of a specific data sample and can be applied to a whole dataset. Our proposed method can achieve a much higher attack ratio than the existing work [40] and enjoys a lower computational cost.

ASNet has two main goals: to detect the necessary neurons and layers, and to compress the existing network. To fulfill the first goal, we require a pretrained model because from Lemmas 3.1, and 3.2, the accuracy of the reduced model will approach that of the original one. For the second task, the pretrained model helps us to get a good estimation for the number of active neurons, a proper layer to cut off, and a good initialization for the active subspace layer and polynomial chaos expansion layer. However, a pretrained model is not required because we can construct ASNet in a heuristic way (as done in most deep neural networks): a reasonable guess for the number of active neurons and cut-off layer, and a random parameter initialization for the pre-model, the active subspace layer, and the polynomial chaos expansion layer.

Acknowledgments. We thank the associate editor and referees for their valuable comments and suggestions. We also thank Prof. Yuxiang Wang at UCSB for providing access to some computing facilities.

REFERENCES

[1] S. Abdoli, L. G. Hafemann, J. Rony, I. B. Ayed, P. Cardinal, and A. L. Koerich, *Universal Adversarial Audio Perturbations*, arXiv preprint, arXiv:1908.03173, 2019.

[2] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, *Net-Trim: Convex pruning of deep neural networks with performance guarantee*, in Proceedings of the Conference on Neural Information Processing Systems, 2017, pp. 3177–3186.

[3] L. Armijo, *Minimization of functions having Lipschitz continuous first partial derivatives*, Pacific J. mathematics, 16 (1966), pp. 1–3.

[4] S. Baluja and I. Fischer, *Adversarial Transformation Networks: Learning to Generate Adversarial Examples*, arXiv preprint, arXiv:1703.09387, 2017.

[5] M. Behjati, S.-M. Moosavi-Dezfooli, M. S. Baghshah, and P. Frossard, *Universal adversarial attacks on text classifiers*, in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2019, pp. 7345–7349.

[6] E. G. Birgin, J. M. Martínez, and M. Raydan, *Nonmonotone spectral projected gradient methods on convex sets*, SIAM J. Optim., 10 (2000), pp. 1196–1211.

[7] H. Cai, L. Zhu, and S. Han, *ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware*, arXiv preprint, arXiv:1812.00332, 2018.

[8] N. Carlini and D. Wagner, *Towards evaluating the robustness of neural networks*, in Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), IEEE, 2017, pp. 39–57.

[9] P. G. Constantine, *Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies*, SIAM, Philadelphia, PA, 2015.

[10] P. G. Constantine, E. Dow, and Q. Wang, *Active subspace methods in theory and practice: Applications to kriging surfaces*, SIAM J. Sci. Comput., 36 (2014), pp. A1500–A1524.

[11] P. G. Constantine, M. Emory, J. Larsson, and G. Iaccarino, *Exploiting active subspaces to quantify uncertainty in the numerical simulation of the HyShot* II *scramjet*, J. Comput. Phys., 302 (2015), pp. 1–20.

[12] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, *Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to* +1 *or* -1, arXiv preprint, arXiv:1602.02830, 2016.

[13] C. Cui and Z. Zhang, *Stochastic collocation with non-Gaussian correlated process variations: Theory, algorithms and applications*, IEEE Trans. Components Packaging Manuf. Tech., 9 (2019), pp. 1362–1375.

[14] C. Cui and Z. Zhang, *High-dimensional uncertainty quantification of electronic and photonic IC with non-gaussian correlated process variations*, IEEE Trans. Comput-Aided Design Integr. Circuits Syst., 39 (2020), pp. 1649–1661.

[15] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, *GXNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework*, Neural Networks, 100 (2018), pp. 49–58.

[16] G. K. Dziugaite, Z. Ghahramani, and D. M. Roy, *A Study of the Effect of JPG Compression on Adversarial Images*, arXiv preprint, arXiv:1608.00853, 2016.

[17] J. Frankle and M. Carbin, *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*, arXiv preprint, arXiv:1803.03635, 2018.

[18] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, *Ultimate Tensorization: Compressing Convolutional and FC Layers Alike*, arXiv preprint, arXiv:1611.03214, 2016.

[19] R. Ge, R. Wang, and H. Zhao, *Mildly Overparametrized Neural Nets Can Memorize Training Data Efficiently*, arXiv preprint, arXiv:1909.11837, 2019.

[20] R. G. Ghanem and P. D. Spanos, *Stochastic finite element method: Response statistics*, in Stochastic Finite Elements: A Spectral Approach, Springer, Cham, 1991, pp. 101–119.

[21] M. Ghashami, E. Liberty, J. M. Phillips, and D. P. Woodruff, *Frequent directions: Simple and deterministic matrix sketching*, SIAM J. Comput., 45 (2016), pp. 1762–1792.

[22] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and Harnessing Adversarial Examples*, arXiv preprint, arXiv:1412.6572, 2014.

[23] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, *Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks*, in Proceedings of the 23rd International Conference on Machine Learning, ACM, 2006, pp. 369–376.

[24] N. Halko, P.-G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288.

[25] S. Han, H. Mao, and W. J. Dally, *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*, arXiv preprint, arXiv:1510.00149, 2015.

[26] C. Hawkins and Z. Zhang, *Bayesian Tensorized Neural Networks with Automatic Rank Selection*, arXiv preprint, arXiv:1905.10478, 2019.

[27] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, *AMC: AutoML for model compression and acceleration on mobile devices*, in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 784–800.

[28] G. Hinton, O. Vinyals, and J. Dean, *Distilling the Knowledge in a Neural Network*, arXiv preprint, arXiv:1503.02531, 2015.

[29] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, in The Collected Works of Wassily Hoeffding, Springer, Cham, 1994, pp. 409–426.

[30] D. W. Hosmer, Jr., S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, John Wiley & Sons, New York, 2013.

[31] I. Jolliffe, *Principal component analysis*, in International Encyclopedia of Statistical Science, Springer, Cham, 2011, pp. 1094–1096.

[32] C. Kanbak, S.-M. Moosavi-Dezfooli, and P. Frossard, *Geometric robustness of deep networks: analysis and improvement*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4441–4449.

[33] V. Khrulkov and I. Oseledets, *Art of singular vectors and universal adversarial perturbations*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8562–8570.

[34] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv preprint, arXiv:1412.6980, 2014.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Proceedings of the Conference on Neural Information Processing Systems, 2012, pp. 1097–1105.

[36] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, *Speeding-up Convolutional Neural Networks Using Fine-Tuned CP-Decomposition*, arXiv preprint, arXiv:1412.6553, 2014.

[37] D. R. Lide, *Handbook of mathematical functions*, in A Century of Excellence in Measurements, Standards, and Technology, CRC Press, Boca Raton, FL, 2018, pp. 135–139.

[38] L. Liu, L. Deng, X. Hu, M. Zhu, G. Li, Y. Ding, and Y. Xie, *Dynamic Sparse Graph for Efficient Deep Learning*, arXiv preprint, arXiv:1810.00859, 2018.

[39] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, *Rethinking the Value of Network Pruning*, arXiv preprint, arXiv:1810.05270, 2018.

[40] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, *Universal adversarial perturbations*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1765–1773.

[41] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, *DeepFool: A simple and accurate method to fool deep neural networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2574–2582.

[42] P. Neekhara, S. Hussain, P. Pandey, S. Dubnov, J. McAuley, and F. Koushanfar, *Universal Adversarial Perturbations for Speech Recognition Systems*, arXiv preprint, arXiv:1905.03828, 2019.

[43] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, *Tensorizing neural networks*, in Proceedings of the Conference on Neural Information Processing Systems, 2015, pp. 442–450.

[44] S. Oymak and M. Soltanolkotabi, *Towards moderate overparameterization: Global convergence guarantees for training shallow neural networks*, IEEE J. Sel. Areas Inform. Therory, 1 (2020), pp. 84–105.

[45] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, *The limitations of deep learning in adversarial settings*, in Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2016, pp. 372–387.

[46] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, *FitNets: Hints for Thin Deep Nets*, arXiv preprint, arXiv:1412.6550, 2014.

[47] H. L. Royden, *Real Analysis*, Macmillan, New York, 2010.

[48] T. M. Russi, *Uncertainty Quantification with Experimental Data and Complex System Models*, PhD thesis, UC Berkeley, Berkeley, CA, 2010.

[49] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*, in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 6655–6659.

[50] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, *Group sparse regularization for deep neural networks*, Neurocomputing, 241 (2017), pp. 81–89.

[51] M. Schmidt, E. Berg, M. Friedlander, and K. Murphy, *Optimizing costly functions with simple constraints: A limited-memory projected quasi-Newton algorithm*, in Proceedings of the International Conference on Artificial Intelligence and Statistics, 2009, pp. 456–463.

[52] A. C. Serban and E. Poll, *Adversarial Examples-A Complete Characterisation of the Phenomenon*, arXiv preprint, arXiv:1810.01185, 2018.

[53] S. Shalev-Shwartz and T. Zhang, *Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization*, in Proceedings of the International Conference on Machine Learning, 2014, pp. 64–72.

[54] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing Properties of Neural Networks*, arXiv preprint, arXiv:1312.6199, (2013).

[55] D. Xiu and G. E. Karniadakis, *Modeling uncertainty in steady state diffusion problems via generalized polynomial chaos*, Comput. Methods Appl. Mech. Engrg., 191 (2002), pp. 4927–4948.

[56] D. Xiu and G. E. Karniadakis, *The Wiener–Askey polynomial chaos for stochastic differential equations*, SIAM J. Sci. Comput., 24 (2002), pp. 619–644.

[57] S. Ye, X. Feng, T. Zhang, X. Ma, S. Lin, Z. Li, K. Xu, W. Wen, S. Liu, J. Tang, M. Fardad, X. Lin, Y. Liu, and Y. Wang, *Progressive DNN Compression: A Key to Achieve Ultra-High Weight Pruning and Quantization Rates Using ADMM*, arXiv preprint, arXiv:1903.09769, 2019.

[58] T. Young, D. Hazarika, S. Poria, and E. Cambria, *Recent trends in deep learning based natural language processing*, IEEE Comput. Intel. Mag., 13 (2018), pp. 55–75.

[59] V. P. Zankin, G. V. Ryzhakov, and I. Oseledets, *Gradient Descent-Based D-Optimal Design for the Least-Squares Polynomial Approximation*, arXiv preprint, arXiv:1806.06631, 2018.