Optimization Landscape and Expressivity of Deep CNNs

Quynh Nguyen [1] **Matthias Hein** [2]

Abstract

We analyze the loss landscape and expressiveness of practical deep convolutional neural networks (CNNs) with shared weights and max pooling layers. We show that such CNNs produce linearly independent features at a "wide" layer which has more neurons than the number of training samples. This condition holds e.g. for the VGG network. Furthermore, we provide for such wide CNNs necessary and sufficient conditions for global minima with zero training error. For the case where the wide layer is followed by a fully connected layer we show that almost every critical point of the empirical loss is a global minimum with zero training error. Our analysis suggests that both depth and width are very important in deep learning. While depth brings more representational power and allows the network to learn high level features, width smoothes the optimization landscape of the loss function in the sense that a sufficiently wide network has a well-behaved loss surface with almost no bad local minima.

1. Introduction

It is well known that the optimization problem for training neural networks can have exponentially many local minima (Auer et al., 1996; Safran & Shamir, 2016) and NP-hardness has been shown in many cases (Blum & Rivest., 1989; Sima, 2002; Livni et al., 2014; Shamir, 2017; Shalev-Shwartz et al., 2017). However, it has been empirically observed (Dauphin et al., 2014; Goodfellow et al., 2015) that the training of state-of-the-art deep CNNs (LeCun et al., 1990; Krizhevsky et al., 2012), which are often overparameterized, is not hampered by suboptimal local minima.

In order to explain the apparent gap between hardness results and practical performance, many interesting theoretical results have been recently developed (Andoni et al., 2014; Sedghi & Anandkumar, 2015; Janzamin et al., 2016; Haeffele & Vidal, 2017; Gautier et al., 2016; Brutzkus & Globerson, 2017; Soltanolkotabi, 2017; Zhong et al., 2017; Tian, 2017; Du et al., 2018) in order to identify conditions under which one can guarantee that local search algorithms like gradient descent converge to the globally optimal solution. However, it turns out that these approaches are either not practical as they require e.g. knowledge about the data generating measure, or a modification of network structure and objective, or they are for quite restricted network structures, mostly one hidden layer networks, and thus are not able to explain the success of deep networks in general. For deep linear networks one has achieved a quite complete picture of the loss surface as it has been shown that every local minimum is a global minimum (Baldi & Hornik, 1988; Kawaguchi, 2016; Freeman & Bruna, 2017; Hardt & Ma, 2017; Yun et al., 2018). By randomizing the nonlinear part of a feedforward network with ReLU activation function and making some additional simplifying assumptions, (Choromanska et al., 2015a) can relate the loss surface of neural networks to a certain spin glass model. In this model the objective of local minima is close to the global optimum and the number of bad local minima decreases quickly with the distance to the global optimum. This is a very interesting result but is based on a number of unrealistic assumptions (Choromanska et al., 2015b). More recently, (Nguyen & Hein, 2017) have analyzed deep fully connected networks with general activation functions and could show that almost every critical point is a global minimum if one layer has more neurons than the number of training points. While this result holds for networks in practice, it requires a quite extensively overparameterized network.

Table 1. The maximum width of all layers in several state-of-the-art CNN architectures compared with the size of ImageNet dataset ($N \approx 1200K$). All numbers are lower bounds on the true width.

CNN ARCHITECTURE	$M = \max_k n_k$	$M > N$
VGG(A-E) (SIMONYAN & ZISSERMAN, 2015)	3000K($k = 1$)	YES
INCEPTIONV3 (SZEGEDY ET AL., 2015B)	1300K($k = 3$)	YES
INCEPTIONV4 (SZEGEDY ET AL., 2016)	1300K($k = 3$)	YES
SQUEEZENET (IANDOLA ET AL., 2016)	1180K($k = 1$)	NO
ENET (PASZKE ET AL., 2016)	1000K($k = 1$)	NO
GOOGLENET (SZEGEDY ET AL., 2015A)	800K($k = 1$)	NO
RESNET (HE ET AL., 2016)	800K($k = 1$)	NO
XCEPTION (CHOLLET, 2016)	700K($k = 1$)	NO

[1]Department of Mathematics and Computer Science, Saarland University, Germany [2]University of Tübingen, Germany. Correspondence to: Quynh Nguyen <quynh@cs.uni-saarland.de>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

In this paper we overcome the restriction of previous work in several ways. This paper is one of the first ones, which analyzes the loss surface of deep CNNs. CNNs are of high practical interest as they learn very useful representations (Zeiler & Fergus, 2014; Mahendran & Vedaldi, 2015; Yosinski et al., 2015) with small number of parameters. We are only aware of (Cohen & Shashua, 2016) who study the expressiveness of CNNs with max-pooling layer and ReLU activation but with rather unrealistic filters (just 1×1) and no shared weights. In our setting we allow as well max pooling and general activation functions. Moreover, we can have an arbitrary number of filters and we study general convolutions as the filters need not be applied to regular structures like 3×3 but can be patch-based where the only condition is that all the patches have the size of the filter. Convolutional layers, fully connected layers and max-pooling layers can be combined in almost arbitrary order. We study in this paper the expressiveness and loss surface of a CNN where one layer is wide, in the sense that it has more neurons than the number of training points. While this assumption sounds at first quite strong, we want to emphasize that the popular VGG (Simonyan & Zisserman, 2015) and Inception networks (Szegedy et al., 2015b; 2016), see Table 1, fulfill this condition. We show that wide CNNs produce linearly independent feature representations at the wide layer and thus are able to fit the training data exactly (universal finite sample expressivity). This is even true with probability one when all the parameters up to the wide layer are chosen randomly[1]. We think that this explains partially the results of (Zhang et al., 2017) where they show experimentally for several CNNs that they are able to fit random labels. Moreover, we provide necessary and sufficient conditions for global minima with zero squared loss and show for a particular class of CNNs that almost all critical points are globally optimal, which to some extent explains why wide CNNs can be optimized so efficiently. All proofs are moved to the appendix due to limited space.

2. Deep Convolutional Neural Networks

We first introduce our notation and definition of CNNs. Let N be the number of training samples and denote by $X = [x_1, \ldots, x_N]^T \in \mathbb{R}^{N \times d}, Y = [y_1, \ldots, y_N]^T \in \mathbb{R}^{N \times m}$ the input resp. output matrix for the training data $(x_i, y_i)_{i=1}^N$, where d is the input dimension and m the number of classes.

Let L be the number of layers of the network, where each layer is either a convolutional, max-pooling or fully connected layer. The layers are indexed from $k = 0, 1, \ldots, L$ which corresponds to input layer, 1st hidden layer, ..., and output layer. Let n_k be the width of layer k and $f_k : \mathbb{R}^d \to \mathbb{R}^{n_k}$ the function that computes for every input its feature vector at layer k.

[1]for any probability measure on the parameter space which has a density with respect to the Lebesgue measure

The convolutional layer consists of a set of patches of equal size where every patch is a subset of neurons from the same layer. Throughout this paper, we assume that the patches of every layer cover the whole layer, *i.e.* every neuron belongs to at least one of the patches, and that there are no patches that contain exactly the same subset of neurons. This means that if one patch covers the whole layer then it must be the only patch of the layer. Let P_k and l_k be the number of patches resp. the size of each patch at layer k for every $0 \leq k < L$. For every input $x \in \mathbb{R}^d$, let $\left\{x^1, \ldots, x^{P_0}\right\} \in \mathbb{R}^{l_0}$ denote the set of patches at the input layer and $\left\{f_k^1(x), \ldots, f_k^{P_k}(x)\right\} \in \mathbb{R}^{l_k}$ the set of patches at layer k. Each filter of the layer consists of the same set of patches. We denote by T_k the number of convolutional filters and by $W_k = [w_k^1, \ldots, w_k^{T_k}] \in \mathbb{R}^{l_{k-1} \times T_k}$ the corresponding parameter matrix of the convolutional layer k for every $1 \leq k < L$. Each column of W_k corresponds to one filter. Furthermore, $b_k \in \mathbb{R}^{n_k}$ denotes the bias vector and $\sigma_k : \mathbb{R} \to \mathbb{R}$ the activation function for each layer. Note that one can use the same activation function for all layers but we use the general form to highlight the role of different layers. In this paper, all functions are applied componentwise, and we denote by $[a]$ the set of integers $\{1, 2, \ldots, a\}$ and by $[a, b]$ the set of integers from a to b.

Definition 2.1 (Convolutional layer) *A layer k is called a convolutional layer if its output $f_k(x) \in \mathbb{R}^{n_k}$ is defined for every $x \in \mathbb{R}^d$ as*

$$f_k(x)_h = \sigma_k\Big(\left\langle w_k^t, f_{k-1}^p(x) \right\rangle + (b_k)_h \Big) \qquad (1)$$

for every $p \in [P_{k-1}], t \in [T_k], h := (p-1)T_k + t$.

The value of each neuron at layer k is computed by first taking the inner product between a filter of layer k and a patch at layer $k-1$, adding the bias and then applying the activation function. The number of neurons at layer k is thus $n_k = T_k P_{k-1}$, which we denote as the width of layer k. Our definition of a convolutional layer is quite general as every patch can be an arbitrary subset of neurons of the same layer and thus covers most of existing variants in practice.

Definition 2.1 includes the fully connected layer as a special case by using $P_{k-1} = 1, l_{k-1} = n_{k-1}, f_{k-1}^1(x) = f_{k-1}(x) \in \mathbb{R}^{n_{k-1}}, T_k = n_k, W_k \in \mathbb{R}^{n_{k-1} \times n_k}, b_k \in \mathbb{R}^{n_k}$. Thus we have only one patch which is the whole feature vector at this layer.

Definition 2.2 (Fully connected layer) *A layer k is called a fully connected layer if its output $f_k(x) \in \mathbb{R}^{n_k}$ is defined for every $x \in \mathbb{R}^d$ as*

$$f_k(x) = \sigma_k\Big(W_k^T f_{k-1}(x) + b_k \Big). \qquad (2)$$

For some results we also allow max-pooling layers.

Definition 2.3 (Max-pooling layer) *A layer k is called a max-pooling layer if its output $f_k(x) \in \mathbb{R}^{n_k}$ is defined for every $x \in \mathbb{R}^d$ and $p \in [P_{k-1}]$ as*

$$f_k(x)_p = \max\left(\left(f_{k-1}^p(x)\right)_1, \ldots, \left(f_{k-1}^p(x)\right)_{l_{k-1}}\right). \quad (3)$$

A max-pooling layer just computes the maximum element of every patch from the previous layer. Since there are P_{k-1} patches at layer $k-1$, the number of output neurons at layer k is $n_k = P_{k-1}$.

Reformulation of Convolutional Layers: For each convolutional or fully connected layer, we denote by $\mathcal{M}_k : \mathbb{R}^{l_{k-1} \times T_k} \to \mathbb{R}^{n_{k-1} \times n_k}$ the linear map that returns for every parameter matrix $W_k \in \mathbb{R}^{l_{k-1} \times T_k}$ the corresponding full weight matrix $U_k = \mathcal{M}_k(W_k) \in \mathbb{R}^{n_{k-1} \times n_k}$. For convolutional layers, U_k can be seen as the counterpart of the weight matrix W_k in fully connected layers. We define $U_k = \mathcal{M}_k(W_k) = W_k$ if layer k is fully connected. Note that the mapping $\mathcal{M}_k$ depends on the patch structure of each convolutional layer k. For example, suppose that layer k has two filters of length 3, that is, $W_k = [w_k^1, w_k^2] = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$, and $n_{k-1} = 5$ and patches given by a 1D-convolution with stride 1 and no padding then:

$$U_k^T = \mathcal{M}_k(W_k)^T = \begin{bmatrix} a & b & c & 0 & 0 \\ d & e & f & 0 & 0 \\ 0 & a & b & c & 0 \\ 0 & d & e & f & 0 \\ 0 & 0 & a & b & c \\ 0 & 0 & d & e & f \end{bmatrix}.$$

The above ordering of the rows of U_k^T of a convolutional layer is determined by (1), in particular, the row index h of U_k^T is calculated as $h = (p-1)T_k + t$, which means for every given patch p one has to loop over all the filters t and compute the corresponding value of the output unit by taking the inner product of the h-th row of U_k^T with the whole feature vector of the previous layer. We assume throughout this paper that that there is no non-linearity at the output layer. By ignoring max-pooling layers for the moment, the feature maps $f_k : \mathbb{R}^d \to \mathbb{R}^{n_k}$ can be written as

$$f_k(x) = \begin{cases} x & k = 0 \\ \sigma_k\big(g_k(x)\big) & 1 \le k \le L-1 \\ g_L(x) & k = L \end{cases}$$

where $g_k : \mathbb{R}^d \to \mathbb{R}^{n_k}$ is the pre-activation function:

$$g_k(x) = U_k^T f_{k-1}(x) + b_k, \quad \forall 1 \le k \le L$$

By stacking the feature vectors of layer k of all training samples, before and after applying the activation function, into a matrix, we define:

$$F_k = [f_k(x_1), \ldots, f_k(x_N)]^T \in \mathbb{R}^{N \times n_k},$$
$$G_k = [g_k(x_1), \ldots, g_k(x_N)]^T \in \mathbb{R}^{N \times n_k}.$$

In this paper, we refer to F_k as the output matrix at layer k. It follows from above that

$$F_k = \begin{cases} X & k = 0 \\ \sigma_k(G_k) & 1 \le k \le L-1 \\ G_L & k = L \end{cases} \quad (4)$$

where $G_k = F_{k-1}U_k + \mathbf{1}_N b_k^T$ for every $1 \le k \le L$.

In this paper, we assume the following general condition on the structure of convolutional layers.

Assumption 2.4 (Convolutional Structure) *For every convolutional layer k, there exists at least one parameter matrix $W_k \in \mathbb{R}^{l_{k-1} \times T_k}$ for which the corresponding weight matrix $U_k = \mathcal{M}_k(W_k) \in \mathbb{R}^{n_{k-1} \times n_k}$ has full rank.*

It is straightforward to see that Assumption 2.4 is satisfied if every neuron belongs to at least one patch and there are no identical patches. As the set of full rank matrices is a dense subset, the following result follows immediately.

Lemma 2.5 *If Assumption 2.4 holds, then for every convolutional layer k, the set of $W_k \in \mathbb{R}^{l_{k-1} \times T_k}$ for which $U_k = \mathcal{M}_k(W_k) \in \mathbb{R}^{n_{k-1} \times n_k}$ does not have full rank has Lebesgue measure zero.*

3. CNN Learn Linearly Independent Features

In this section, we show that a class of standard CNN architectures with convolutional layers, fully connected layers and max-pooling layers plus standard activation functions like ReLU, sigmoid, softplus, etc are able to learn linearly independent features at every wide hidden layer if it has more neurons than the number of training samples. Our assumption on training data is the following.

Assumption 3.1 (Training data) *The patches of different training samples are non-identical, that is, $x_i^p \neq x_j^q$ for every $p, q \in [P_0], i, j \in [N], i \neq j$.*

Assumption 3.1 is quite weak, especially if the size of the input patches is large. If the assumption does not hold, one can add a small perturbation to the training samples: $\{x_1 + \epsilon_1, \ldots, x_N + \epsilon_N\}$. The set of $\{\epsilon_i\}_{i=1}^N$ where Assumption 3.1 is not fulfilled for the new dataset has measure zero. Moreover, $\{\epsilon_i\}_{i=1}^N$ can be chosen arbitrarily small so that the influence of the perturbation is negligible. Our main assumptions on the activation function of the hidden layers are the following.

Assumption 3.2 (Activation function) *The activation function σ is continuous, non-constant, and satisfies one of the following conditions:*

- *There exist $\mu_+, \mu_- \in \mathbb{R}$ s.t. $\lim_{t\to-\infty} \sigma_k(t) = \mu_-$ and $\lim_{t\to\infty} \sigma_k(t) = \mu_+$ and $\mu_+\mu_- = 0$*
- *There exist $\rho_1, \rho_2, \rho_3, \rho_4 \in \mathbb{R}_+$ s.t. $|\sigma(t)| \leq \rho_1 e^{\rho_2 t}$ for $t < 0$ and $|\sigma(t)| \leq \rho_3 t + \rho_4$ for $t \geq 0$*

Assumption 3.2 covers several standard activation functions.

Lemma 3.3 *The following activation functions satisfy Assumption 3.2:*

- *ReLU: $\sigma(t) = \max(0, t)$*
- *Sigmoid: $\sigma(t) = \frac{1}{1+e^{-t}}$*
- *Softplus: $\sigma_\alpha(t) = \frac{1}{\alpha}\ln(1 + e^{\alpha t})$ for some $\alpha > 0$*

It is known that the softplus function is a smooth approximation of ReLU. In particular, it holds that:

$$\lim_{\alpha\to\infty} \sigma_\alpha(t) = \lim_{\alpha\to\infty} \frac{1}{\alpha}\ln(1 + e^{\alpha t}) = \max(0, t). \quad (5)$$

The first main result of this paper is the following.

Theorem 3.4 (Linearly Independent Features) *Let Assumption 3.1 hold for the training sample. Consider a deep CNN architecture for which there exists some layer $1 \leq k \leq L - 1$ such that*

1. *Layer 1 and layer k are convolutional or fully connected while all the other layers can be convolutional, fully connected or max-pooling*
2. *The width of layer k is larger than the number of training samples, $n_k = T_k P_{k-1} \geq N$*
3. *$(\sigma_1, \ldots, \sigma_k)$ satisfy Assumption 3.2*

Then there exists a set of parameters of the first k layers $(W_l, b_l)_{l=1}^k$ such that the set of feature vectors $\{f_k(x_1), \ldots, f_k(x_N)\}$ are linearly independent. Moreover, $(W_l, b_l)_{l=1}^k$ can be chosen in such a way that all the weight matrices $U_l = \mathcal{M}_l(W_l)$ have full rank for every $1 \leq l \leq k$.

Theorem 3.4 implies that a large class of CNNs employed in practice with standard activation functions like ReLU, sigmoid or softplus can produce linearly independent features at any hidden layer if its width is larger than the size of training set. Figure 1 shows an example of a CNN architecture that satisfies the conditions of Theorem 3.4 at the first convolutional layer. Note that if a set of vectors is linearly independent then they are also linearly separable. In this sense, Theorem 3.4 suggests that CNNs can produce linearly separable features at every wide hidden layer.

Linear separability in neural networks has been recently studied by (An et al., 2015), where the authors show that a two-hidden-layer fully connected network with ReLU activation function can transform any training set to be linearly separable while approximately preserving the distances of the training data at the output layer. Compared to (An et al., 2015) our Theorem 3.4 is derived for CNNs with a wider range of activation functions. Moreover, our result shows even linear independence of features which is stronger than linear separability. Recently, (Nguyen & Hein, 2017) have shown a similar result for fully connected networks and analytic activation functions.

We want to stress that, in contrast to fully connected networks, for CNNs the condition $n_k \geq N$ of Theorem 3.4 does not imply that the network has a huge number of parameters as the layers k and $k + 1$ can be chosen to be convolutional. In particular, the condition $n_k = T_k P_{k-1} \geq N$ can be fulfilled by increasing the number of filters T_k or by using a large number of patches P_{k-1} (however P_{k-1} is upper bounded by n_k), which is however only possible if l_{k-1} is small as otherwise our condition on the patches cannot be fulfilled. In total the CNN has only $l_{k-1}T_k + l_k T_{k+1}$ parameters versus $n_k(n_{k-1} + n_{k+1})$ for the fully connected network from and to layer k. Interestingly, the VGG-Net (Simonyan & Zisserman, 2015), where in the first layer small 3×3 filters and stride 1 is used, fulfills for ImageNet the condition $n_k \geq N$ for $k = 1$, as well as the Inception networks (Szegedy et al., 2015b; 2016), see Table 1.

One might ask now how difficult it is to find such parameters which generate linearly independent features at a hidden layer? Our next result shows that once analytic[2] activation functions, *e.g.* sigmoid or softplus, are used at the first k hidden layers of the network, the linear independence of features at layer k holds with probability 1 even if one draws the parameters of the first k layers $(W_l, b_l)^k$ randomly for any probability measure on the parameter space which has a density with respect to the Lebesgue measure.

Theorem 3.5 *Let Assumption 3.1 hold for the training samples. Consider a deep CNN for which there exists some layer $1 \leq k \leq L - 1$ such that*

1. *Every layer from 1 to k is convolutional or fully connected*
2. *The width of layer k is larger than number of training*

[2]A function $\sigma : \mathbb{R} \to \mathbb{R}$ is real analytic if its Taylor series about x_0 converges to $\sigma(x_0)$ on some neighborhood of x_0 for every $x_0 \in \mathbb{R}$ (Krantz & Parks, 2002).

samples, that is, $n_k = T_k P_{k-1} \geq N$

3. *$(\sigma_1, \ldots, \sigma_k)$ are real analytic functions and satisfy Assumption 3.2.*

*Then the set of parameters of the first k layers $(W_l, b_l)_{l=1}^k$ for which the set of feature vectors $\{f_k(x_1), \ldots, f_k(x_N)\}$ are **not** linearly independent has Lebesgue measure **zero**.*

Theorem 3.5 is a much stronger statement than Theorem 3.4, as it shows that for almost all weight configurations one gets linearly independent features at the wide layer. While Theorem 3.5 does not hold for the ReLU activation function as it is not an analytic function, we note again that one can approximate the ReLU function arbitrarily well using the softplus function (see 5), which is analytic function for any $\alpha > 0$ and thus Theorem 3.5 applies. It is an open question if the result holds also for the ReLU activation function itself. The condition $n_k \geq N$ is not very restrictive as several state-of-the art CNNs , see Table 1, fulfill the condition. Furthermore, we would like to stress that Theorem 3.5 is *not* true for deep linear networks. The reason is simply that the rank of a product of matrices can at most be the minimal rank among all the matrices. The nonlinearity of the activation function is thus critical (note that the identity activation function, $\sigma(x) = x$, does not fulfill Assumption 3.2).

To illustrate Theorem 3.5 we plot the rank of the feature matrices of the network in Figure 1. We use the MNIST dataset with $N = 60000$ training and 10000 test samples. We add small Gaussian noise $\mathcal{N}(0, 10^{-5})$ to every training sample so that Assumption 3.1 is fulfilled. We then vary the number of convolutional filters T_1 of the first layer from 10 to 100 and train the corresponding network with squared loss and sigmoid activation function using Adam (Kingma & Ba, 2015) and decaying learning rate for 2000 epochs. In Table 2 we show the smallest singular value of the feature matrices together with the corresponding training loss, training and test error. If number of convolutional filters is large enough (*i.e.* $T_1 \geq 89$), one has $n_1 = 26 \times 26 \times T_1 \geq N = 60000$, and the second condition of Theorem 3.5 is satisfied for $k = 1$. Table 2 shows that the feature matrices F_1 have full rank in all cases (and F_3 in almost all cases), in particular for $T_1 \geq 89$ as shown in Theorem 3.5. As expected when the feature maps of the training samples are linearly independent after the first layer (F_1 has rank 60000 for $T \geq 89$) the training error is zero and the training loss is close to zero (the GPU uses single precision). However, as linear independence is stronger than linear separability one can achieve already for $T < 89$ zero training error.

It is interesting to note that Theorem 3.5 explains previous empirical observations. In particular, (Czarnecki et al., 2017) have shown empirically that linear separability is often obtained already in the first few hidden layers of the trained networks. This is done by attaching a linear classifier probe (Alain & Bengio, 2016) to every hidden layer in the network after training the whole network with backpropagation. The fact that Theorem 3.5 holds even if the parameters of the bottom layers up to the wide layer k are chosen randomly is also in line with recent empirical observations for CNN architectures that one has little loss in performance if the weights of the initial layers are chosen randomly without training (Jarrett et al., 2009; Saxe et al., 2011; Yosinski et al., 2014).

As a corollary of Theorem 3.4 we get the following universal finite sample expressivity for CNNs. In particular, a deep CNN with scalar output can perfectly fit any scalar-valued function for a finite number of inputs if the width of the last hidden layer is larger than the number of training samples.

Corollary 3.6 (Universal Finite Sample Expressivity) *Let Assumption 3.1 hold for the training samples. Consider a standard CNN with scalar output which satisfies the conditions of Theorem 3.4 at the last hidden layer $k = L - 1$. Let $f_L : \mathbb{R}^d \to \mathbb{R}$ be the output of the network given as*

$$f_L(x) = \sum_{j=1}^{n_{L-1}} \lambda_j f_{(L-1)j}(x) \quad \forall x \in \mathbb{R}^d$$

where $\lambda \in \mathbb{R}^{n_{L-1}}$ is the weight vector of the last layer. Then for every target $y \in \mathbb{R}^N$, there exists $\left\{\lambda, (W_l, b_l)_{l=1}^{L-1}\right\}$ so that it holds $f_L(x_i) = y_i$ for every $i \in [N]$.

The expressivity of neural networks has been well-studied, in particular in the universal approximation theorems for one hidden layer networks (Cybenko, 1989; Hornik et al., 1989). Recently, many results have shown why deep networks are superior to shallow networks in terms of expressiveness (Delalleau & Bengio, 2011; Telgarsky, 2016; 2015; Eldan & Shamir, 2016; Safran & Shamir, 2017; Yarotsky, 2016; Poggio et al., 2016; Liang & Srikant, 2017; Mhaskar & Poggio, 2016; Montufar et al., 2014; Pascanu et al., 2014; Raghu et al., 2017) While most of these results are derived for fully connected networks, it seems that (Cohen & Shashua, 2016) are the first ones who study expressivity of CNNs. In particular, they show that CNNs with max-pooling and ReLU units are universal in the sense that they can approximate any given function if the size of the networks is unlimited. However, the number of convolutional filters in this result has to grow exponentially with the number of patches and they do not allow shared weights in their result, which is a standard feature of CNNs. Corollary 3.6 shows universal finite sample expressivity, instead of universal function approximation, even for $L = 2$ and $k = 1$, that is a single convolutional layer network can perfectly fit the training data as long as the number of hidden units is larger than the number of training samples.

For fully connected networks, universal finite sample ex-

Table 2. The smallest singular value $\sigma_{\min}(F_1)$ of the feature matrix F_1 of the first convolutional layer (similar $\sigma_{\min}(F_3)$ for the feature matrix F_3 of the second convolutional layer) of the trained network in Figure 1 are shown for varying number of convolutional filters T_1. The rank of a matrix $A \in \mathbb{R}^{m\times n}$ is estimated (see Chapter 2.6.1 in (Press, 2007)) by computing the singular value decomposition of A and counting the singular values which exceed the threshold $\frac{1}{2}\sqrt{m+n+1}\,\sigma_{\max}(A)\epsilon$, where ϵ is machine precision. For all filter sizes the feature matrices F_1 have full rank. Zero training error is attained for $T_1 \geq 30$.

T_1	SIZE(F_1)	*rank*(F_1)	$\sigma_{\text{MIN}}(F_1)$	SIZE(F_3)	*rank*(F_3)	$\sigma_{\text{MIN}}(F_3)$	LOSS($\times 10^{-5}$)	TRAIN ERROR	TEST ERROR
10	60000×6760	**6760**	3.7×10^{-6}	60000×2880	**2880**	2.0×10^{-2}	2.4	8	151
20	60000×13520	**13520**	2.2×10^{-6}	60000×2880	**2880**	7.0×10^{-4}	1.2	1	132
30	60000×20280	**20280**	1.5×10^{-6}	60000×2880	**2880**	2.4×10^{-4}	0.24	**0**	174
40	60000×27040	**27040**	2.0×10^{-6}	60000×2880	**2880**	2.2×10^{-3}	0.62	**0**	124
50	60000×33800	**33800**	1.3×10^{-6}	60000×2880	**2880**	3.9×10^{-5}	0.02	**0**	143
60	60000×40560	**40560**	1.1×10^{-6}	60000×2880	**2880**	4.0×10^{-5}	0.57	**0**	141
70	60000×47320	**47320**	7.5×10^{-7}	60000×2880	**2880**	7.1×10^{-3}	0.12	**0**	120
80	60000×54080	**54080**	5.4×10^{-7}	60000×2880	2875	4.9×10^{-18}	0.11	**0**	140
89	60000×60164	**60000**	2.0×10^{-8}	60000×2880	**2880**	8.9×10^{-10}	0.35	**0**	117
100	60000×67600	**60000**	1.1×10^{-6}	60000×2880	2856	8.5×10^{-27}	0.04	**0**	139

pressivity has been studied by (Zhang et al., 2017; Nguyen & Hein, 2017; Hardt & Ma, 2017). It is shown that a single hidden layer fully connected network with N hidden units can express any training set of size N. While the number of training parameters of a single hidden layer CNN with N hidden units and scalar output is just $2N + T_1 l_0$, where T_1 is the number of convolutional filters and l_0 is the size of each filter, it is $Nd + 2N$ for fully connected networks. If we set the width of the hidden layer of the CNN as $n_1 = T_1 P_0 = N$ in order to fulfill the condition of Corollary 3.6, then the number of training parameters of the CNN becomes $2N + Nl_0/P_0$, which is less than $3N$ if $l_0 \leq P_0$ compared to $(d+2)N$ for the fully connected case. In practice one almost always has $l_0 \leq P_0$ as l_0 is typically a small integer and P_0 is on the order of the dimension of the input. Thus, the number of parameters to achieve universal finite sample expressivity is for CNNs significantly smaller than for fully connected networks.

Obviously, in practice it is most important that the network generalizes rather than just fitting the training data. By using shared weights and sparsity structure, CNNs seem to implicitly regularize the model to achieve good generalization performance. Thus even though they can fit also random labels or noise (Zhang et al., 2017) due to the universal finite sample expressivity shown in Corollary 3.6, they seem still to be able to generalize well (Zhang et al., 2017).

4. Optimization Landscape of Deep CNNs

In this section, we restrict our analysis to the use of least squares loss. However, as we show later that the network can produce exactly the target output (*i.e.* $F_L = Y$) for some choice of parameters, all our results can also be extended to any other loss function where the global minimum is attained at $F_L = Y$, for instance the squared Hinge-loss analyzed in (Nguyen & Hein, 2017). Let $\mathcal{P}$ denote the space of all parameters of the network. The final training objective $\Phi : \mathcal{P} \to \mathbb{R}$ is given as

$$\Phi\Big((W_l, b_l)_{l=1}^L\Big) = \frac{1}{2}\left\|F_L - Y\right\|_F^2 \tag{6}$$

where F_L is defined as in (4), which is also the same as

$$F_L = \sigma_{L-1}(\ldots\sigma_1(XU_1 + \mathbf{1}_N b_1^T)\ldots)U_L + \mathbf{1}_N b_L^T,$$

where $U_l = \mathcal{M}_l(W_l)$ for every $1 \leq l \leq L$. We require the following assumptions on the architecture of CNN.

Assumption 4.1 (CNN Architecture) *Every layer in the network is a convolutional layer or fully connected layer and the output layer is fully connected. Moreover, there exists some hidden layer $1 \leq k \leq L-1$ such that the following holds:*

- *The width of layer k is larger than number of training samples, that is, $n_k = T_k P_{k-1} \geq N$*
- *All the activation functions of the hidden layers $(\sigma_1, \ldots, \sigma_{L-1})$ satisfy Assumption 3.2*
- *$(\sigma_{k+1}, \ldots, \sigma_{L-1})$ are strictly increasing or strictly decreasing, and differentiable*
- *The network is pyramidal from layer $k+1$ till the output layer, that is, $n_{k+1} \geq \ldots \geq n_L$*

A typical example that satisfies Assumption 4.1 with $k = 1$ can be found in Figure 1 where one disregards max-pooling layers and uses *e.g.* sigmoid or softplus activation function.

In the following, let us define for every $1 \leq k \leq L-1$ the subset $S_k \subseteq \mathcal{P}$ of the parameter space such that

$$S_k := \left\{(W_l, b_l)_{l=1}^L \;\middle|\; F_k, U_{k+2}, \ldots, U_L \text{have full rank}\right\}.$$

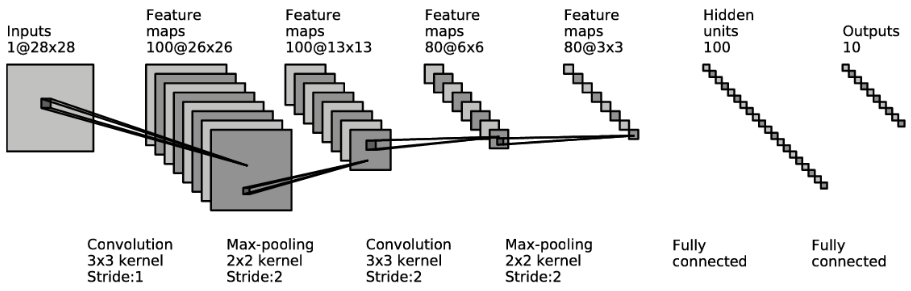

Figure 1. An example of CNN for a given training set of size $N \leq 100 \times 26 \times 26 = 67600$. The width of each layer is $d = n_0 = 784, n_1 = 67600, n_2 = 16900, n_3 = 2880, n_4 = 720, n_5 = 100, n_6 = m = 10$. One can see that $n_1 \geq N$ and the network has pyramidal structure from layer 2 till the output layer, that is, $n_2 \geq \ldots \geq n_6$.

The set S_k is the set of parameters where the feature matrix at layer k and all the weight matrices from layer $k+2$ till the output layer have full rank. In the following, we examine conditions for global optimality in S_k. It is important to note that S_k covers almost the whole parameter space under an additional mild condition on the activation function.

Lemma 4.2 *Let Assumption 3.1 hold for the training samples and a deep CNN satisfy Assumption 4.1 for some layer $1 \leq k \leq L-1$. If the activation functions of the first k layers $(\sigma_1, \ldots, \sigma_k)$ are real analytic, then the complementary set $\mathcal{P} \setminus S_k$ has Lebesgue measure zero.*

In the next key lemma, we bound the objective function in terms of its gradient magnitude w.r.t. the weight matrix of layer k for which $n_k \geq N$. For every matrix $A \in \mathbb{R}^{m \times n}$, let $\sigma_{\min}(A)$ and $\sigma_{\max}(A)$ denote the smallest and largest singular value of A. Let $\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$, $\|A\|_{\min} := \min_{i,j} |A_{ij}|$ and $\|A\|_{\max} := \max_{i,j} |A_{ij}|$. From (4), and (6), it follows that Φ can be seen as a function of $(U_l, b_l)_{l=1}^L$, and thus we can use $\nabla_{U_k}\Phi$. If layer k is fully connected then $U_k = \mathcal{M}_k(W_k) = W_k$ and thus $\nabla_{U_k}\Phi = \nabla_{W_k}\Phi$. Otherwise, if layer k is convolutional then we note that $\nabla_{U_k}\Phi$ is "not" the true gradient of the training objective because U_k is not the true optimization parameter but W_k. In this case, the true gradient of Φ w.r.t. to the true parameter matrix W_k which consists of convolutional filters can be computed via the chain rule as

$$\frac{\partial \Phi}{\partial (W_k)_{rs}} = \sum_{i,j} \frac{\partial \Phi}{\partial (U_k)_{ij}} \frac{\partial (U_k)_{ij}}{\partial (W_k)_{rs}}$$

Please note that even though we write the partial derivatives with respect to the matrix elements, $\nabla_{W_k}\Phi$ resp. $\nabla_{U_k}\Phi$ are the matrices of the same dimension as W_k resp. U_k in the following.

Lemma 4.3 *Consider a deep CNN satisfying Assumption 4.1 for some hidden layer $1 \leq k \leq L-1$. Then it holds*

$$\begin{aligned}&\left\|\nabla_{U_{k+1}}\Phi\right\|_F\\ \geq&\sigma_{\mathit{min}}(F_k)\Big(\prod_{l=k+1}^{L-1} \sigma_{\mathit{min}}(U_{l+1}) \left\|\sigma_l'(G_l)\right\|_{\mathit{min}}\Big) \left\|F_L - Y\right\|_F\end{aligned}$$

and

$$\begin{aligned}&\left\|\nabla_{U_{k+1}}\Phi\right\|_F\\ \leq&\sigma_{\mathit{max}}(F_k)\Big(\prod_{l=k+1}^{L-1} \sigma_{\mathit{max}}(U_{l+1}) \left\|\sigma_l'(G_l)\right\|_{\mathit{max}}\Big) \left\|F_L - Y\right\|_F .\end{aligned}$$

Our next main result is motivated by the fact that empirically when training over-parameterized neural networks with shared weights and sparsity structure like CNNs, there seem to be no problems with sub-optimal local minima. In many cases, even when training labels are completely random, local search algorithms like stochastic gradient descent can converge to a solution with almost zero training error (Zhang et al., 2017). To understand better this phenomenon, we first characterize in the following Theorem 4.4 the set of points in parameter space with zero loss, and then analyze in Theorem 4.5 the loss surface for a special case of the network. We emphasize that our results hold for standard deep CNNs with convolutional layers with shared weights and fully connected layers.

Theorem 4.4 (Conditions for Zero Training Error) *Let Assumption 3.1 hold for the training sample and suppose that the CNN architecture satisfies Assumption 4.1 for some hidden layer $1 \leq k \leq L-1$. Let $\Phi : \mathcal{P} \to \mathbb{R}$ be defined as in (6). Given any point $(W_l, b_l)_{l=1}^L \in S_k$. Then it holds that $\Phi\Big((W_l, b_l)_{l=1}^L\Big) = 0$ if and only if $\nabla_{U_{k+1}}\Phi\Big|_{(W_l,b_l)_{l=1}^L} = 0$.*

Proof: If $\Phi\Big((W_l, b_l)_{l=1}^L\Big) = 0$ then it follows from the upper bound of Lemma 4.3 that $\nabla_{U_{k+1}}\Phi = 0$. For reverse direction, one has $(W_l, b_l)_{l=1}^L \in S_k$ and thus

$\mathit{rank}(F_k) = N$ and U_l has full rank for every $l \in [k+2, L]$. Thus it holds $\sigma_{\min}(F_k) > 0$ and $\sigma_{\min}(U_l) > 0$ for every $l \in [k+2, L]$. Moreover, $(\sigma_{k+1}, \ldots, \sigma_{L-1})$ have non-zero derivative by Assumption 4.1 and thus $\left\|\sigma_l'(G_l)\right\|_{\min} > 0$ for every $l \in [k+1, L-1]$. This combined with the lower bound in Lemma 4.3 leads to $\Phi\Big(W_l, b_l)_{l=1}^L\Big) = \left\|F_L - Y\right\|_F = 0$. $\square$

Lemma 4.2 shows that the set of points which are not covered by Theorem 4.4 has measure zero if the first k layers have analytic activation functions. The necessary and sufficient condition of Theorem 4.4 is rather intuitive as it requires the gradient of the training objective to vanish w.r.t. the full weight matrix of layer $k+1$ regardless of the architecture of this layer. It turns out that if layer $k+1$ is fully connected, then this condition is always satisfied at a critical point, in which case we obtain that every critical point in S_k is a global minimum with exact zero training error. This is shown in the next Theorem 4.5, where we consider a classification task with m classes, $Z \in \mathbb{R}^{m\times m}$ is the full rank class encoding matrix e.g. the identity matrix and (X, Y) the training sample such that $Y_{i:} = Z_{j:}$ whenever the training sample x_i belongs to class j for every $i \in [N], j \in [m]$.

Theorem 4.5 (Loss Surface of CNNs) *Let (X, Y, Z) be a training set for which Assumption 3.1 holds, the CNN architecture satisfies Assumption 4.1 for some hidden layer $1 \leq k \leq L-1$, and layer $k+1$ is fully connected. Let $\Phi : \mathcal{P} \to \mathbb{R}$ be defined as in (6). Then the following holds*

- *Every critical point $(W_l, b_l)_{l=1}^L \in S_k$ is a global minimum with $\Phi\Big((W_l, b_l)_{l=1}^L\Big) = 0$*
- *There exist infinitely many global minima $(W_l, b_l)_{l=1}^L \in S_k$ with $\Phi\Big((W_l, b_l)_{l=1}^L\Big) = 0$*

Theorem 4.5 shows that the loss surface for this type of CNNs has a rather simple structure in the sense that every critical point in S_k must be a global minimum with zero training error. Note that if the activation functions up to layer k are analytic, the complement of S_k has measure zero (see Lemma 4.2). For those critical points lying outside S_k, it must hold that either one of the weight matrices $\{U_{k+2}, \ldots, U_L\}$ has low rank or the set of feature vectors at layer k is not linearly independent (*i.e.* F_k has low rank). Obviously, some of these critical points can also be global minima, but we conjecture that they cannot be suboptimal local minima due to the following reasons. First, it seems unlikely that a critical point with a low rank weight matrix is a suboptimal local minimum as this would imply that all possible full rank perturbations of the current solution must have larger/equal objective value. However, there is no term in the loss function which favors low rank solutions. Even for linear networks, it has been shown by (Baldi & Hornik, 1988) that all the critical points with low rank weight matrices have to be saddle points and thus cannot be suboptimal local minima. Second, a similar argument applies to the case where one has a critical point outside S_k such that the features are not linearly independent. In particular, any neighborhood of such a critical point contains points which have linearly independent features at layer k, from which it is easy to reach zero loss if one fixes the parameters of the first k layers and optimizes the loss w.r.t. the remaining ones. This implies that every small neighborhood of the critical point should contain points from which there exists a descent path that leads to a global minimum with zero loss, which contradicts the fact that the critical point is a suboptimal local minimum. In summary, if there are critical points lying outside the set S_k, then it is very “unlikely” that these are suboptimal local minima but rather also global minima, saddle points or local maxima.

It remains an interesting open problem if the result of Theorem 4.5 can be transferred to the case where layer $k+1$ is also convolutional. In any case whether layer $k+1$ is fully connected or not, one might assume that a solution with zero training error still exists as it is usually the case for practical over-parameterized networks. However, note that Theorem 4.4 shows that at those points where the loss is zero, the gradient of Φ w.r.t. U_{k+1} must be zero as well.

An interesting special case of Theorem 4.5 is when the network is fully connected in which case all the results of Theorem 4.5 hold without any modifications. This can be seen as a formal proof for the implicit assumption used in the recent work (Nguyen & Hein, 2017) that there exists a global minimum with absolute zero training error for the class of fully connected, deep and wide networks.

5. Conclusion

We have analyzed the expressiveness and loss surface of CNNs in realistic and practically relevant settings. As state-of-the-art networks fulfill exactly or approximately the condition to have a sufficiently wide convolutional layer, we think that our results help to understand why current CNNs can be trained so effectively. It would be interesting to discuss the loss surface for cross-entropy loss, which currently does not fit into our analysis as the global minimum does not exist when the data is linearly separable.

Acknowledgements

The authors would like to thank the reviewers for their helpful comments on the paper and Maksym Andriushchenko for helping us to set up the experiment on the rank of features.

References

Alain, G. and Bengio, Y. Understanding intermediate layers using linear classifier probes. In *ICLR Workshop*, 2016.

An, S., Boussaid, F., and Bennamoun, M. How can deep rectifier networks achieve linear separability and preserve distances? In *ICML*, 2015.

Andoni, A., Panigrahy, R., Valiant, G., and Zhang, L. Learning polynomials with neural networks. In *ICML*, 2014.

Auer, P., Herbster, M., and Warmuth, M. K. Exponentially many local minima for single neurons. In *NIPS*, 1996.

Baldi, P. and Hornik, K. Neural networks and principle component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58, 1988.

Blum, A. and Rivest., R. L. Training a 3-node neural network is np-complete. In *NIPS*, 1989.

Brutzkus, A. and Globerson, A. Globally optimal gradient descent for a convnet with gaussian inputs. In *ICML*, 2017.

Chollet, F. Xception: Deep learning with depthwise separable convolutions, 2016. arXiv:1610.02357.

Choromanska, A., Hena, M., Mathieu, M., Arous, G. B., and LeCun, Y. The loss surfaces of multilayer networks. In *AISTATS*, 2015a.

Choromanska, A., LeCun, Y., and Arous, G. B. Open problem: The landscape of the loss surfaces of multilayer networks. *COLT*, 2015b.

Cohen, N. and Shashua, A. Convolutional rectifier networks as generalized tensor decompositions. In *ICML*, 2016.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

Czarnecki, W. M., Swirszcz, G., Jaderberg, M., Osindero, S., Vinyals, O., and Kavukcuoglu, K. Understanding synthetic gradients and decoupled neural interfaces. In *ICML*, 2017.

Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*, 2014.

Delalleau, O. and Bengio, Y. Shallow vs. deep sum-product networks. In *NIPS*, 2011.

Du, S. S., Lee, J. D., and Tian, Y. When is a convolutional filter easy to learn? In *ICLR*, 2018.

Eldan, R. and Shamir, O. The power of depth for feedforward neural networks. In *COLT*, 2016.

Freeman, C. D. and Bruna, J. Topology and geometry of half-rectified network optimization. In *ICLR*, 2017.

Gautier, A., Nguyen, Q., and Hein, M. Globally optimal training of generalized polynomial neural networks with nonlinear spectral methods. In *NIPS*, 2016.

Goodfellow, I. J., Vinyals, O., and Saxe, A. M. Qualitatively characterizing neural network optimization problems. In *ICLR*, 2015.

Haeffele, B. D. and Vidal, R. Global optimality in neural network training. In *CVPR*, 2017.

Hardt, M. and Ma, T. Identity matters in deep learning. In *ICLR*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5mb model size, 2016. arXiv:1602.07360.

Janzamin, M., Sedghi, H., and Anandkumar, A. Beating the perils of non-convexity: Guaranteed training of neural networks using tensor methods. *arXiv:1506.08473*, 2016.

Jarrett, K., Kavukcuoglu, K., and LeCun, Y. What is the best multi-stage architecture for object recognition? In *CVPR*, 2009.

Kawaguchi, K. Deep learning without poor local minima. In *NIPS*, 2016.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Krantz, S. G. and Parks, H. R. *A Primer of Real Analytic Functions*. Birkhäuser, Boston, second edition, 2002.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., and Jackel, L.D. Handwritten digit recognition with a back-propagation network. In *NIPS*, 1990.

Liang, S. and Srikant, R. Why deep neural networks for function approximation? In *ICLR*, 2017.

Livni, R., Shalev-Shwartz, S., and Shamir, O. On the computational efficiency of training neural networks. In *NIPS*, 2014.

Mahendran, A. and Vedaldi, A. Understanding deep image representations by inverting them. In *CVPR*, 2015.

Mhaskar, H. and Poggio, T. Deep vs. shallow networks : An approximation theory perspective, 2016. arXiv:1608.03287.

Montufar, G., Pascanu, R., Cho, K., and Bengio, Y. On the number of linear regions of deep neural networks. In *NIPS*, 2014.

Nguyen, Q. and Hein, M. The loss surface of deep and wide neural networks. In *ICML*, 2017.

Pascanu, R., Montufar, G., and Bengio, Y. On the number of response regions of deep feedforward networks with piecewise linear activations. In *ICLR*, 2014.

Paszke, A., Chaurasia, A., Kim, S., and Culurciello, E. Enet: A deep neural network architecture for real-time semantic segmentation, 2016. arXiv:1606.02147.

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., and Liao, Q. Why and when can deep – but not shallow – networks avoid the curse of dimensionality: a review, 2016. arXiv:1611.00740.

Press, W. H. *Numerical Recipes: The art of scientific computing*. Cambridge University Press, 2007.

Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Sohl-Dickstein, J. On the expressive power of deep neural networks. In *ICML*, 2017.

Safran, I. and Shamir, O. On the quality of the initial basin in overspecified networks. In *ICML*, 2016.

Safran, I. and Shamir, O. Depth-width tradeoffs in approximating natural functions with neural networks. In *ICML*, 2017.

Saxe, A., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y. On random weights and unsupervised feature learning. In *ICML*, 2011.

Sedghi, H. and Anandkumar, A. Provable methods for training neural networks with sparse connectivity. In *ICLR Workshop*, 2015.

Shalev-Shwartz, S., Shamir, O., and Shammah, S. Failures of gradient-based deep learning. In *ICML*, 2017.

Shamir, O. Distribution-specific hardness of learning neural networks, 2017. arXiv:1609.01037.

Sima, J. Training a single sigmoidal neuron is hard. *Neural Computation*, 14:2709–2728, 2002.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Soltanolkotabi, M. Learning relus via gradient descent. In *NIPS*, 2017.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *CVPR*, 2015a.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision, 2015b. arXiv:1512.00567.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016. arXiv:1602.07261.

Telgarsky, M. Representation benefits of deep feedforward networks, 2015. arXiv:1509.08101v2.

Telgarsky, M. Benefits of depth in neural networks. In *COLT*, 2016.

Tian, Y. An analytical formula of population gradient for two-layered relu network and its applications in convergence and critical point analysis. In *ICML*, 2017.

Yarotsky, D. Error bounds for approximations with deep relu networks, 2016. arXiv:1610.01145.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In *NIPS*, 2014.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. Understanding neural networks through deep visualization. In *ICML*, 2015.

Yun, C., Sra, S., and Jadbabaie, A. Global optimality conditions for deep neural networks. In *ICLR*, 2018.

Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, Oriol. Understanding deep learning requires re-thinking generalization. In *ICLR*, 2017.

Zhong, K., Song, Z., Jain, P., Bartlett, P., and Dhillon, I. Recovery guarantees for one-hidden-layer neural networks. In *ICML*, 2017.