Analysis of Deep Neural Networks with the Extended Data Jacobian Matrix

Shengjie Wang WANGSJ@CS.WASHINGTON.EDU
Abdel-rahman Mohamed ASAMIR@MICROSOFT.COM
Rich Caruana RCARUANA@MICROSOFT.COM
Jeff Bilmes BILMES@UW.EDU
Matthai Plilipose MATTHAIP@MICROSOFT.COM
Matthew Richardson MATTRI@MICROSOFT.COM
Krzysztof Geras K.J.GERAS@SMS.ED.AC.UK
Gregor Urban GURBAN@UCI.EDU
Ozlem Aslan OZLEM@CS.UALBERTA.CA

Abstract

Deep neural networks have achieved great success on a variety of machine learning tasks. There are many fundamental and open questions yet to be answered, however. We introduce the *Extended Data Jacobian Matrix* (EDJM) as an architecture-independent tool to analyze neural networks at the manifold of interest. The spectrum of the EDJM is found to be highly correlated with the complexity of the learned functions. After studying the effect of dropout, ensembles, and model distillation using EDJM, we propose a novel spectral regularization method, which improves network performance.

1. Introduction

Deep Learning has achieved significant success in the past few years on many challenging tasks (Krizhevsky et al., 2012; Hinton et al., 2012; Simonyan & Zisserman, 2014; Bahdanau et al., 2014; Mnih et al., 2015; Silver et al., 2016; Sutskever et al., 2014) , but there is still much work needed to understand why deep architectures are able to learn such effective representations.

One proposal (Pascanu et al., 2013; Montufar et al., 2014) argues that deep neural networks with rectified units are able to separate the input space into exponentially more linear response regions than shallow networks given the same number of computational units, thus enabling deep networks to learn highly complex and structured functions. Empirical results in (Romero et al., 2014) show that deep but thin networks can mimic the function learned by shallower (but still deep) and wide networks. Even for deep linear networks, (Saxe et al., 2013) found that they exhibit non-linear training dynamics similar to non-linear deep neural networks. Although deep linear networks are equivalent to shallow linear networks in representational power, their training dynamics are rich in mathematical structure that alter key aspects of the network such as training time, and the effects of pre-training, random initialization, and generalization. Better understanding of the trajectory of the functions learned by different architectures during and at the end of training is important for understanding why deep learning is so effective, how different architectures affect what is learned, and for developing networks with better regularization and generalization abilities. For example, in (Neyshabur et al., 2015), understanding and enforcing scale invariance in ReLU networks led to developing a more resilient update rule than standard SGD for networks with unbalanced weights.

We started this work motivated by trying to understand what qualities deep networks possess that shallow networks do not, and also how convolutional neural networks differ from feed-forward ones. Recently, model compression and distillation (Ba & Caruana, 2014; Hinton et al., 2015) show that it is possible to train compact models to approximate the functions learned by more complex models. Empirically, on some datasets, shallow networks can be improved dramatically to the point of even matching the best deep network by learning from the soft labels generated by a large and deep model or an ensemble of such models. Such improvement may result from the extra information provided by the soft labels which preserves the relative confidence of different outputs learned by the bigger model. The distillation results posed yet another interesting question: in what way does the more complex, "teacher", network alter the function learned by the less complex, "student", network compared to a similar network learning without the benefit of a teacher?

In this paper, conditioned on a set of data points that define

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

the manifold of interest, we propose the Extended Data Jacobian Matrix (EDJM) as an analysis tool for neural networks. By studying the spectrum of EDJM, which we believe is highly correlated with the complexity of the functions learned by networks, we can compare networks with different depths, architectures, and training techniques in a unified way. First, we introduce the Extended Data Jacobian Matrix for feed-forward and convolution neural networks. Then we show the characteristics of the spectrum of the EDJM for the best performing networks over multiple datasets. We also show in section 4, that different methods such as model compression and dropout increases the same measure of the spectrum of the EDJM. Finally, motivated by these observations, we propose a regularization technique for improving performance of deep neural networks.

2. Data Jacobian Matrix

Given a deep neural network consisting of m *linear* layers, the entire network is a series of matrix multiplies that can be represented merely by a single matrix:

$$W_{net} = W_m W_{m-1} \dots W_1 \tag{1}$$

Suppose we are given a data set D, and $|D| = n$, where each data point $(x, y) \in D$ consists of input features x of d_{in} dimensions, and an output label y of d_{out} dimensions. Using the linear network described above, we have, for (x_i, y_i):

$$\hat{y_i} = W_m W_{m-1} \dots W_1 x_i, \quad \text{or} \quad \hat{y_i} = W_{net} x_i, \tag{2}$$

where $\hat{y_i}$ is the predicted label of x_i using the network.

Now we introduce non-linearity into the linear network. Without loss of generality, we use Rectified Linear Units (ReLU), which sets negative entries in a vector to 0 while preserving the positive entries (We will show how to generalize to other types of activation functions later):

$$\phi_{ReLU}(z) = \max(0, z) \tag{3}$$

If we apply ReLU to all the linear layers except for the last layer, we have the following network:

$$\hat{y_i} = W_m \phi_{ReLU}(W_{m-1} \phi_{ReLU}(\dots \phi_{ReLU}(W_1 x_i))) \tag{4}$$

2.1. ReLU Network as a Collection of Linear Systems

Suppose for input x_i, the output at layer l is h_l^i, then $h_l^i = \phi_{ReLU}(W_l h_{l-1}^i)$. If $(W_l h_{l-1}^i)[k] < 0$, then $h_l^i[k] = 0$ according to the ReLU function, which is equivalent to setting the k th row of W_l to be all 0, while if $(W_l h_{l-1}^i)[k] \geq 0$, the row is left exactly the same. Therefore, we can eliminate the non-linear ReLU functions by modifying certain rows of weight matrices to 0 while keeping the rest, and we end up with a linear system for data point x_i:

$$\hat{y}_i = W_m \hat{W}_{m-1}^i \hat{W}_{m-2}^i \dots \hat{W}_1^i x_i \tag{5}$$

$$\hat{y}_i = W_{net}^i x_i \tag{6}$$

where:

$$\hat{W}_l^i[a, b] = \begin{cases} W_l[a, b] & \text{if } h_l^i[a] > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{7}$$

Note that W_{net}^i is x_i specific, as the modification that transforms W_l into $\hat{W}_l^i$ depends on the values of h_l^i, and therefore is based on the input features x_i. For each individual data point in a finite data set, we can construct a linear model W_{net}^i of dimensions $d_{out} \times d_{in}$ in such manner, and that generates the same output as the original neural network consisting of arbitrary number of layers and hidden units.

Clearly, for different data points x_i and x_j, different linear systems will most likely be constructed. However, for an x_j that is close enough to x_i, the output of the first layer h_1^i and h_1^j could have the same patterns of ReLU activations In other words, $(W_1 x_i)[k] < 0 \iff (W_1 x_j)[k] < 0 \; \forall k$, so in this case, the modified weight matrices are the same for both data points. If such pattern is carried over multiple layers, the constructed linear systems could be the same. Therefore, for input data points in a small region around x_i, the constructed linear system for x_i remains unchanged, so that the ReLU network is piece-wise linear.

2.2. Data Jacobian Matrix

For dataset $D = \{(x_i, y_i)\}$, where x is of dimension d_{in} and y is of dimension d_{out}, and a deep neural network $\hat{y}_i = f_m(\phi_{m-1}(f_{m-1}(\phi_{m-2}(\dots \phi_1(f_1(x_i))))))$, where f is a linear operation, and ϕ is a differentiable non-linear function, we denote the *Data Jacobian Matrix* for input features x_i as:

$$DJM_\theta(x_i) = \frac{\partial \hat{y}_i}{\partial x_i} = \frac{\partial \hat{y}_i}{\partial h_{m-1}^i} \frac{\partial h_{m-1}^i}{\partial h_{m-2}^i} \dots \frac{\partial h_1^i}{\partial x_i} \tag{8}$$

$$= \frac{\partial f_m(h_{m-1}^i)}{\partial h_{m-1}^i} \frac{\partial h_{m-1}^i}{\partial f_{m-1}(h_{m-2}^i)} \dots \frac{\partial f_1(x_i)}{\partial x_i}. \tag{9}$$

The subscript θ in the DJM denotes that depends on the neural network parameters θ. For ReLU feed-forward network in specific, we have:

$$DJM_\theta(x_i) = W_m \frac{\partial h_{m-1}^i}{\partial W_{m-1} h_{m-2}^i} W_{m-1} \frac{\partial h_{m-2}^i}{\partial W_{m-2} h_{m-3}^i} \dots W_1$$

In addition, under ReLU non-linear function, we have:

$$\frac{\partial h_{l+1}^i[a]}{\partial (W_{l+1} h_l^i)[b]} = \begin{cases} 1 \text{ if } a = b \text{ and } (W_{l+1} h_l^i)[b] \geq 0 \\ 0 \text{ otherwise} \end{cases}.$$

It is easy to see that:

$$DJM_\theta(x_i) = W_m \hat{W}_{m-1}^i \hat{W}_{m-2}^i \dots \hat{W}_1^i = W_{net}^i \tag{10}$$

$$\hat{y_i} = DJM_\theta(x_i) x_i, \tag{11}$$

For ReLU feed-forward network, the Data Jacobian Matrix for x_i is equivalent to the per-data point linear system for x_i (Eq 11). Such equivalence also holds for ReLU convolutional networks with max/average pooling layers. Convolution, as a linear transformation, can be written in the form of matrix multiplication, and thus no different from the feed-forward layers when calculating the gradients. Average pooling is a special case of convolution, which can be viewed as a matrix multiplication as well. Max pooling outputs the highest value out of a certain receptive field, which is equivalent to removing certain rows of the weight matrix based on the input values. Similar to the ReLU function, when calculating the gradients, elements in the removed rows receive 0 gradients and therefore the equivalence holds.

Conceptually, the Data Jacobian Matrix is the gradient of the outputs of the neural network with respect to the inputs. Suppose the neural network is applied to a classification task, ReLU networks are characterized with piece-wise linear classification boundaries, which collide with the gradients. For non-linear functions other than ReLU (e.g. sigmoid, and tangent), the gradients are linear approximations to the classification boundaries.

3. Extended Data Jacobian Matrix

For simplicity, suppose $d_{out} = 1$ (i.e. the task is binary classification or a single value regression), for data point x_i, $DJM_\theta(x_i)$ is a single vector. Stacking such vectors across different data points in X with $|X| = n$, we get a matrix of dimension $n \times d_{in}$. We denote such a matrix as the *Extended Data Jacobian Matrix* (EDJM). In the general case, for every dimension of the output, we can construct such Extended Data Jacobian Matrix, so that we have d_{out} EDJMs all of which are of dimension $n \times d_{in}$.

The Extended Data Jacobian Matrix is a collection of linear systems across data points, allowing us to study and compare various neural networks conditioned on certain set of data. For ReLU networks in particular, EDJM is equivalent to the neural network for the input dataset. For the rest of the paper, we will focus on ReLU networks for the equivalence we get, yet all the analysis described in the following applies to non-ReLU networks as well.

An EDJM has fixed dimension $n \times d_{in}$ for a given data set regardless of the network structure, thus allowing us to compare different networks (e.g., deep vs. shallow, or convolutional vs. feed-forward) through the same lens.

The spectrum, or distribution of singular values, of EDJM reflects the principle components of the space spanned by different linear systems for different data points. We choose to use matrix factorization over each EDJM instead of tensor factorization on the tensor formed by d_{out} EDJMs for the following reasons: 1) tensor factorization in general is NP-hard and there is identifiability issue with tensor factorization; 2) the computational cost for tensor factorization in our experiment setting can be intractable; 3) matrix factorization of EDJMs can be used to investigate and compare different linear systems that map inputs to different output classes. A spectrum with one single component suggests that the linear system for certain output dimension is the same for all data points, making the neural network very simple, and indeed linear for that output dimension. The other extreme is when the spectrum of EDJM is uniform across all components meaning that the space spanned by the data points' linear systems is extremely complex and the underlying neural network behaves dramatically different for different data points. The distribution of singular values of the EDJM, therefore, naturally reflects the "complexity" of the function learned by the neural network. For the rest of this paper, we will compare many neural networks with different architectures to demonstrate the connection to the spectrum of the EDJM.

4. Empirical Analysis with EDJMs

In this section, we will show that the EDJM spectrum correlates well with our natural notion of model complexity and more advanced training methods, as well as being predictive of model accuracy.

Experiments in this paper are conducted on three different datasets: MNIST for hand-written digit recognition, CIFAR-10 for image recognition, and TIMIT for phone recognition. MNIST consists of 60000 training data points, out of which we randomly extract 10000 data points as the validation set, and 10000 testing data points. Each data point is a single channel image of size $(1, 28, 28)$ and thus has 784 dimensional features. CIFAR-10 consists of 50000 training images and 10000 testing images. Similar to MNIST, we extract 10000 out of the training dataset as a validation set. Each image in the CIFAR-10 dataset is of size $(3, 32, 32)$, which is of dimension 3072. Both MNSIT and CIFAR-10 has 10 output classes, so 10 Extended Data Jacobian Matrices can be constructed. For MNIST, the dimensions of the EDJM are $n \times 784$, and for CIFAR-10, the dimensions the EDJM are $n \times 3072$, where the value of n depends on the subset of data selected. For validation set, $n = 10000$ for both MNIST and CIFAR-10. The TIMIT corpus consists of a 462 speaker training set, a 50 speaker validation set, and a 24 speaker test set. 15 frames are grouped together as inputs where each frame contains 40 log mel filterbank coefficients plus energy along with their first and second temporal derivatives.

We use stochastic gradient descent with momentum for training all the following reported models. Learning rates gets halved if the performance does not improve over a succession of 5 epochs on the validation set. No regularization/batch normalization is applied if not specified. The

reported models are all selected by grid search for best performance to cover a broad range for each parameter in order to ensure a fair comparison between models.

4.1. EDJM for Feed-forward Networks

We compare EDJM of neural nets with various depths (# layers = $1, 2, 3, 4$) while we either fix the total number of parameters, or the total number of hidden units. Fixing the total number of parameters constrains the size and computational costs of the network. Moreover, for ReLU networks, fixing the number of hidden units restricts the upper bound on the number of different DJMs that can be generated. Specifically, for ReLU networks with H hidden units, at most 2^H different DJMs can be generated, as the activation/deactivation of ReLU on any hidden unit may contribute to construct a different linear system. Also note that such upper bound is irrespective of the number of layers.

Plotted in Figure 1 is the normalized spectrum (normalized by the max singular value, so the curves start with 1.0 on the left) of EDJMs for different feed-forward networks conditioned on the validation set of MNIST and CIFAR-10, and averaged over 10 output dimensions. As we apply softmax to the output of the networks for classification tasks, which may result in significantly different scales for linear systems of different networks, we report the normalized spectrum. We observe same patterns for training set, validation set and testing set. We choose to present the validation set results as 1) a natural selection of the conditioning dataset for EDJMs is the data utilized during the training process; 2) the accuracy associated with each curve on the validation set is more informative than the training set; 3) analysis of the validation set results can be beneficial to tuning the hyper-parameters of training, which should never be done on the testing set. For all the following comparisons of spectra, we always report the validation set results.

From Figure 1, we observe a clear trend that EDJMs of deeper networks tend to have more dominating singular values. To quantify the plotted shape in one simple metric, we propose the following score:

$$score(S_\theta(X, j)) = \sum_{\sigma \in S_\theta(X,j), \sigma > \epsilon} \frac{\sigma}{\max(S_\theta(X, j))}, \quad (12)$$

where $S_\theta(X, j)$ denotes the singular values for EDJM conditioned on data input X and jth output. Conceptually, the score is the sum over normalized singular values excluding the small ones. From Figure 1 we observe that the singular values decrease at a super-linear rate, and therefore there are lot of small singular values, whose corresponding components are more likely to be noise for our analysis. On the other hand, the proposed score captures the relative power of the important components. Higher score suggests that there are more important components embedded in the given EDJM, or the underlying neural network represents a more complex set of linear systems. The value of ϵ is chosen to be 90 percentile of the singular values (we keep the top 10% singular values since the rest 90% values are small and likely to be noise). The scores averaged over 10 output dimensions for different neural network structures are reported in Figure 1 (C) and (F), which coincide with the trend observed from the spectra.

For either fixed number of hidden units or fixed number of parameters, deeper networks have higher normalized singular values for the major components of the spectrum. The depth of the neural network acts like a prior on the spectrum of the EDJM: deeper neural networks are likely to be trained to generate a more complex set of linear systems conditioned on the dataset of interest, which is consistent with the common belief that more complex functions are learned with deep neural networks than shallow neural networks.

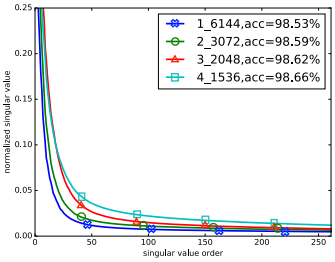
4.2. EDJM for Convolutional Networks

In Figure 2, we show a comparison between the best performing feed-forward network with convolutional networks that have comparable number of parameters. For both datasets, large gaps between the spectrum of convolutional and feed-forward networks are observed, suggesting that the learned functions for the convolution networks are tremendously more complex than the feed-forward ones.

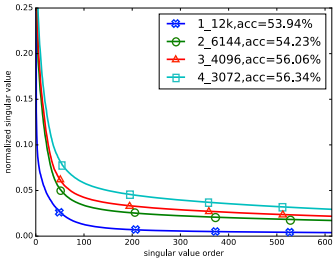
Our contention is that the reason behind the large gaps is in the difference between the number of hidden units of the feed-forward and the convolutional networks. Suppose, for every filter, the output size is the same as the input size (i.e. we do not lose the boundary region when performing convolution). The total number of output units is therefore the number of filters $\times$ the input size. For CIFAR-10, suppose we have 64 filters as our first layer, we then end up with $32 * 32 * 64 = 65536$ hidden units. As stated above, the total number of hidden units puts a restriction on the upper-bound of the number of independent linear systems that can be constructed through the ReLU network. Therefore, convolution networks have a much higher upper-bound, which may result in more complex systems.

4.3. EDJM for Model Compression

So far, we have seen that EDJM scores vary with neural network architecture. In the rest of the paper we will show that the scores also vary with the training technique utilized. Model compression or distillation (Ba & Caruana, 2014; Hinton et al., 2015), focuses on training a compact model to approximate the function learned by a more complex model. Empirically, for certain datasets, a shallow (but not too small) network can be improved dramatically, even matching the best deep network, by learning from the soft labels generated by a large and complex model, or alternatively an ensemble of such models. Such improvements may result from the extra information provided by



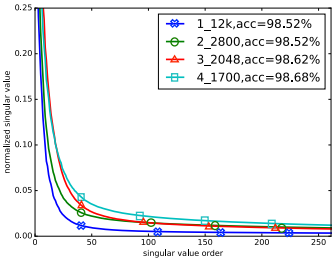
(A) MNIST, Fixed # Hidden Units



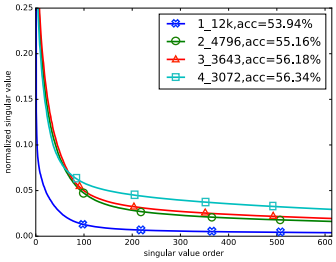
(B) CIFAR-10, Fixed # hidden units

# layers	MNIST score / acc	CIFAR-10 score / acc
1	4.25 / 98.53%	5.99 / 53.94%
2	5.49 / 98.59%	13.94 / 54.23%
3	7.35 / 98.62%	16.58 / 56.06%
4	7.70 / 98.66%	21.40 / 56.34%

(C) Scores for Fixed # Hidden Units (Eq 12)



(D) MNIST, Fixed # Parameters



(E) CIFAR-10, Fixed # Parameters

# layers	MNIST score / acc	CIFAR-10 score / acc
1	4.04 / 98.52%	5.99 / 53.94%
2	5.55 / 98.52%	18.64 / 55.16%
3	7.55 / 98.62%	20.39 / 56.18%
4	7.65 / 98.68%	21.40 / 56.34%

(F) Scores for Fixed # Parameters

Figure 1. Spectra of EDJMs of feed-forward networks with various number of layers. Validation set accuracy is reported in the legends of each figure and in the tables (D) and (E) (same for following figures and tables). (A) Spectra of EDJMs for networks with 1 - 4 layers and fixed number of total hidden units equal to 6k on MNIST. Each spectrum consists of singular values normalized by the largest singular value (i.e., the curve starts at 1.0 on y-axis), sorted in decreasing order, and then averaged over 10 output classes. We set the maximum value of y-axis to be 0.25 for the purpose of better visual display. (B) Spectra of EDJMs for networks with 1 - 4 layers and fixed number of total hidden units equal to 12k on CIFAR-10. (C) Scores for the spectra plotted in (A) and (B). (D) Spectra of EDJMs for networks with 1 - 4 layers and fixed number of parameters equal to 10 million on MNIST. (E) Spectra of EDJMs for networks with 1 - 4 layers and fixed number of parameters equal to 46 million on CIFAR-10. (F) Scores of the spectra plotted in (D) and (E).

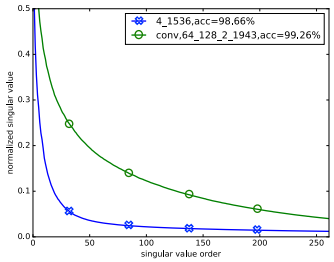
the soft labels, referred as "dark knowledge" in (Hinton et al., 2014), which preserves the relative confidence of different outputs learned by the complex model.

We analyze the EDJM for models trained with the model compression framework to further illustrate EDJM's ability measure the relative complexity of models trained in very different ways, as well as to try to better understand the model compression process. Specifically, we compare the EDJM for five models on the CIFAR-10 dataset (the performance gap between the complex and simple models on MNIST without distillation is too small to be interesting). The five models are as follows:

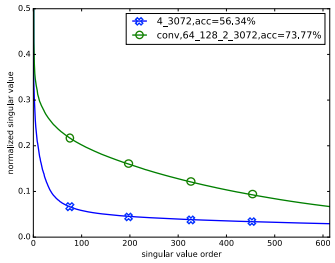
1) Convolutional Teacher Model: A deep convolution network of structure: 64 filter 3 * 3 conv - 64 filter 3 * 3 conv - 2 * 2 max pooling - 128 filter 3 * 3 - 128 filter 3 * 3 - 2 * 2 max pooling - 256 filter 3 * 3 - 256 filter 3 * 3 - 256 filter 3 * 3 - 256 filter 3 * 3 - 2 * 2 max pooling - 2 * 1024 feed-forward layer, and trained with hard 0-1 labels. 2) Convolutional Student Model: A shallow convolution network of structure: 128 filter 5 * 5 - 2 * 2 max pooling - 800 linear layer - 1 * 5000 feed-forward layer, with soft labels provided by the convolution teacher model. 3) Convolutional Shallow Model: Same structure as the convolution student model, but trained on the original hard 0-1 labels. 4) Feed-forward Student Model: A shallow feed-forward network of structure: 1200 linear layer - 1 * 30k feed-forward layer, with soft labels provided by the convolution teacher model. 5) Feed-forward Shallow Model: Same structure as the feed-forward student model, but trained on the original hard 0-1 labels.

The linear bottleneck layer in the models above are utilized to reduce the computational cost, which is also applied in (Ba & Caruana, 2014).

Figure 3 shows the spectrum of the EDJMs for the models described above. The very complex and deep teacher model achieves significantly better performance than the other models as well as highest normalized singular values for the major components. The student models, which benefit from the soft labels provided by the teacher models, get better performance and higher normalized singular values than the shallow models with the same architecture trained on the 0-1 hard targets. Once more, we observe gaps in major component singular values that correlate with the performance gaps between convolution networks and feed-forward networks. It appears that the deep teacher model is able to learn the most complex model, and that the student models trained to mimic the teacher model are able to learn more complex functions than those learned by shallow models of the same architecture that were trained on the original 0-1 hard targets.



(A) MNIST, Conv. v.s. Feed-forward

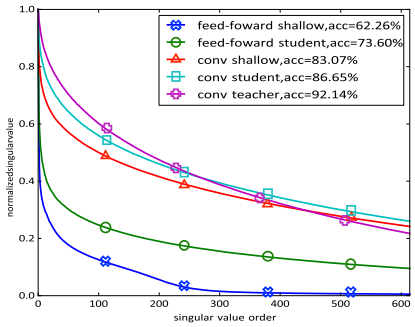


(B) CIFAR-10, Conv. v.s. Feed-forward

Model	MNIST score / acc	CIFAR-10 score / acc
Feed-for.	7.70 / 98.66%	21.40 / 56.34%
Conv.	21.55 / 99.26%	59.65 / 73.77%

(C) Scores for Conv. v.s. Feed-forward

Figure 2. Spectra of EDJMs of convolution networks compared to feed-forward networks. (A) Spectra of EDJMs of a 4 * 1536 feed-forward network compared with a 64 filter 3 * 3 - 2 * 2 max pooling - 128 filter 3 * 3 - 2 * 2 max pooling - 2 * 1943 network on MNIST dataset. (B) Spectra of EDJMs of a 4 * 3072 feed-forward network compared with a 64 filter 3 * 3 - 2 * 2 max pooling - 128 filter 3 * 3 - 2 * 2 max pooling - 2 * 3072 feed-forward network. (C) Scores for the spectra plotted in (A) and (B).



(A) CIFAR-10, Model Compression

Model	Score	Acc
Feed-forward shallow	33.7	62.3%
Feed-forward student	73.6	73.6%
Convolutional shallow	147.8	83.1%
Convolutional student	162.7	86.7%
Convolutional teacher	172.7	92.1%

(B) Scores for Model Compression Results

Figure 3. Effects of model compression training on spectra of EDJMs. (A) shows the spectra of EDJMs for different models in model compression training on CIFAR-10. (B) Scores for (A).

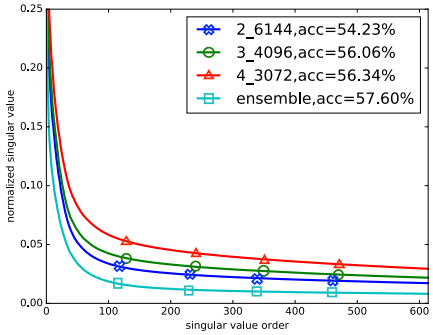
4.4. EDJM for Networks with Dropout

Dropout (Srivastava et al., 2014) is a widely-used technique that disentangles the dependencies among neural network units by randomly deactivating each neural unit with certain probability. As neural units are dropped out randomly for each mini-batch, dropout effectively provides an efficient way of approximately combining a large collections of neural networks with different structures, and thus often improves the performance of the neural networks.

Figure 4 shows that on both MNIST and CIFAR-10, and two different network structures, the dropout networks consistently have higher values on the major components of EDJMs as well as higher performance.

The increase of the major components singular values for dropout networks may result from the independence over hidden units created by dropout. As neural units are dropped-out at random, the correlation among weights is reduced, forcing rows of weight matrices to be more independent of each other. We observe from the EDJMs, which are products of individual weight matrices, that the normalized singular values for major components are higher.

4.5. EDJM for an Ensemble of Networks



(A) CIFAR-10, Ensemble of Networks

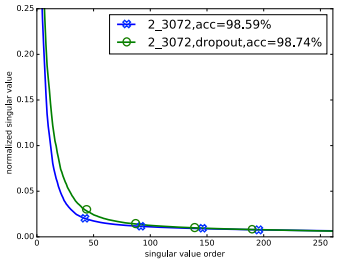
Model	Score	Acc
Feed-forward 2 layer	13.94	54.23%
Feed-forward 3 layer	16.58	56.06%
Feed-forward 4 layer	21.40	56.34%
Ensemble	8.75	57.60%

(B) Scores for Ensemble Results

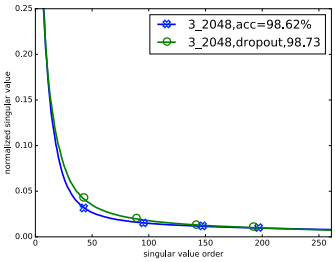
Figure 5. Spectra of EDJMs for ensemble of networks. (A) shows the spectra of 3 different feed-forward networks and their ensemble on CIFAR-10. (B) Scores for spectra plotted in (A).

Figure 5 shows the normalized singular values of the EDJM for three DNNs trained on CIFAR-10, and for an ensemble of those DNNs. As shown before, deeper DNNs are more accurate: the 4-DNN has 56.34% accuracy, the 3-DNN has 56.05%, and the 2-DNN has only 54.23%. Also, the normalized singular value scores correlate with the accuracy of the individual models: the 4-DNN has the highest score and the 2-DNN the lowest score.

But what about the ensemble? As expected it has higher accuracy than the individual models (57.60%), but its nor-



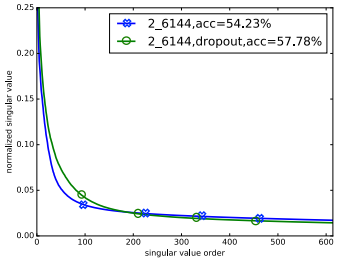
(A) MNIST Dropout 2 * 3k Feed-forward Net



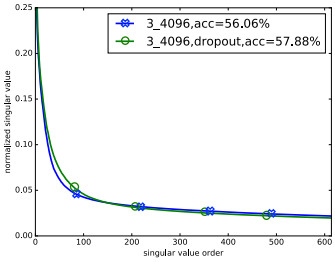
(B) MNIST Dropout 3 * 2k Feed-forward Net

Model	MNIST score / acc
2 layer	5.49 / 98.59%
2 layer dropout	7.15 / 98.74%
3 layer	7.55 / 98.62%
3 layer dropout	8.19 / 98.73%

(C) MNIST Scores for Dropout v.s. Non-dropout



(D) CIFAR-10 Dropout 2 * 6k Feed-forward Net



(E) CIFAR-10 Dropout 3 * 4k Feed-forward Net

Model	CIFAR-10 score / acc
2 layer	13.94 / 54.23%
2 layer dropout	16.33 / 57.78%
3 layer	16.58 / 56.06%
3 layer dropout	17.50 / 57.88%

(F) CIFAR-10 Scores for Dropout v.s. Non-dropout

Figure 4. Effects of dropout on the spectra of EDJMs. (A) and (B) shows the spectra of EDJMs for models of two network structures feed-forward 2 * 3k and 3 * 2k respectively on MNIST. (C) and (D) shows the spectra of EDJMs for models of two network structures feed-forward 2 * 6k and 3 * 4k respectively on CIFAR-10. (E) Table of scores for the spectra plotted in (A), (B), (C) and (D).

malized score is less than the score of even the 2-DNN. This is exactly what we should see if the normalized singular value score is a measure of the relative complexity of the learned functions. Averaging ensembles increase accuracy by reducing variance — an ensemble is *lower* complexity than the individual models contained in it. Averaging suppresses the parts of the learned functions where the models disagree, while emphasizing the regions where the models agree. The net result is that the strongest singular values in the ensemble increase relative to the individual models, while the weaker singular values decrease, both of which lower the score. This result for different model types such as 2-DNNs, 3-DNNs, 4-DNNs and an ensemble of these DNNs provides evidence that the EDJM can be used as an architecture-independent measure of the relative complexity of learned functions.

5. Boosting Performance with EDJMs

All the results shown above always find that the better performing models tend to have higher normalized singular values on the major components of EDJMs. To further investigate this and take advantage of the predictive properties of the EDJM for the success in neural network training, we introduce a regularizer on the singular values of EDJMs. We wish to encourage the major singular values of the EDJM to be high relative to the largest singular value, and this can be done by utilizing the following term:

$$r(X, \lambda) = -\lambda \sum_{j=0}^{d_{out-1}} \sum_{\sigma \in S_\theta(X,j)} \log \sigma, \qquad (13)$$

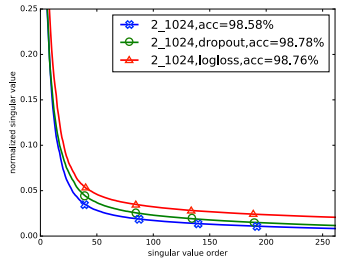
where λ is a hyper-parameter for the regularization term.

Recall that $S_\theta(X, j)$ denotes all the singular for EDJM conditioned on data input X and jth output. The proposed regularizer is the sum of the log of the singular values of EDJM, which enforces the singular values to be all high due to the diminishing return property of the log function.

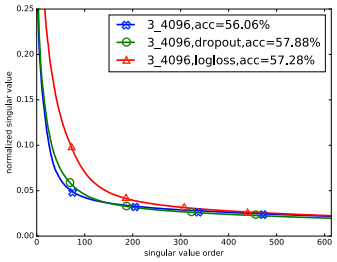
While $r(X, \lambda)$ is differentiable, the computational cost is extremely expensive for training neural networks. This is because: (1) X can be as large as the training set to get an accurate estimate of the singular values; (2) even for approximation, where X is as small as a mini-batch, the extra cost to run singular value decomposition (required to calculate the gradients) of EDJMs every mini-batch is appreciable; and (3) unlike standard back-propagation, the gradient with respect to a certain weight matrix depends on all weight matrices before and after the layer of interest.

To utilize our observations about the spectrum of EDJMs in a more tractable manner, we instead place a regularization term on the singular values of each of the layer weights: $r_{layer}(W_l, \lambda) = -\lambda \sum_{\sigma \in S_{layer}(W_l)} \log \sigma$, where $S_{layer}(W_l)$ denotes the singular values of W_l.

As observed in Section 4.4, dropout effectively boosts the performance as well as increasing the normalized singular values of the major components of EDJMs by modifying the output of every layer. Accordingly, we expect that putting $r_{layer}(W_l, \lambda)$ on the layer weights will affect singular values of EDJMs. Suppose we consider the ReLU function as a stochastic selector on the rows of the weight matrix, by keeping certain rows while "wiping out" the others by setting them to be zero. If the rows of



(A) MNIST Spec. Regularization



(B) CIFAR-10 Spec. Regularization

Model	MNIST score / acc	CIFAR-10 score / acc
Feed-for.	7.00 / 98.58%	16.58 / 56.06%
Dropout	7.51 / 98.78%	17.50 / 57.88%
Spec. Reg.	8.99 / 98.76%	26.44 / 57.28%

(C) Scores for Spec. Regularization

Figure 6. The effect of layer weight spectrum regularizer on spectra of EDJMs. (A) Spectra of baseline feed-forward network, dropout network and spectrum regularized network with same network structure (feed-forward 2 * 1k) on MNIST. (B) Spectra of baseline feed-forward network, dropout network and spectrum regularized network with same network structure (feed-forward 3 * 4k) on CIFAR-10. (C) Table of scores for the curved plotted in (A) and (B).

the weight matrix are already quite dependent, or there is only one dominating singular value for the weight matrix, two subsets of the rows selected by ReLU also tend to be dependent on each other. Conversely, two subsets of independent rows also tend to be independent. For a one hidden layer network, the independence among the modified weight matrices is a good indicator to the independence of rows of EDJMs, which is a sign of high normalized singular values of major components. Such intuition can be carried over to deeper networks as well, as we show in our empirical results. Though such approach is an approximation and not mathematically related to $r(X, \lambda)$, we will show that $r_{layer}(W_l, \lambda)$ can be easily implemented and works well in practice

To incorporate $r_{layer}(W_l, \lambda)$ into fully-connected layers, we introduce SVD layers. For input h_{l-1}, instead of a single matrix multiplication $W_l h_{l-1}$, the SVD layer consists of a series of 3 matrices and thus we have $U_l diag(S_l) V_l h_{l-1}$, where $U_l diag(S_l) V_l = SVD(W_l)$, so that U_l and V_l are both orthonormal, S_l contains the singular values of W_l, and $diag(*)$ represents the diagonal matrix with the input vector on the diagonal. In terms of network structure, replacing the ordinary fully-connected layer with dimensions $|h_l| \times |h_{l-1}|$, the SVD layer consists of 3 sub-layers: 1) U layer: a fully connected layer with dimensions $|h_l| \times \min(|h_l|, |h_{l-1}|)$; 2) S layer: a layer with a weight vector of length $\min(|h_l|, |h_{l-1}|)$; 3) V layer: a fully connected layer with dimensions $\min(|h_l|, |h_{l-1}|) \times |h_{l-1}|$. Since we decompose one fully-connected layer into three linear layers, the computational cost for training is about 3 times more than ordinary. However, for inference, we can merge the three linear matrices back together, and there is no extra cost. As the S layer corresponds to the singular values of W_l, we can regularize S layer directly and easily: $r_{layer}(W_l = U_l diag(S_l) V_l, \lambda) = -\lambda \sum_{\sigma \in S_l} \log \sigma$.

To enforce U_l and V_l to be ortho-normal during training, we first update U_l, S_l and V_l to be U_l', S_l', and V_l' according to the gradients, and then set $U_l diag(S_l) V_l = SVD(U_l' diag(S_l') V_l')$. To save the extra computation introduced by the singular value decomposition operation, such projection can be done less frequently. In practice, we do this projection every epoch, which seems to work well.

Dataset	Structure	Feed-for.	Dropout	Spec. Reg.
MNIST	2 * 1k	98.48%	98.70%	98.66%
CIFAR-10	3 * 4k	56.93%	57.50%	57.43%
TIMIT	3 * 2k	77.77%	78.10%	78.78%

Table 1. Test set accuracy (phone accuracy for TIMIT) on MNIST, CIFAR-10 and TIMIT datasets, comparing the baseline feed-forward network with either adding dropout or the spectrum regularization on singular values to the same network structure.

Results shown in Table 1 shows improved performance on test set with SVD layers on MNIST, CIFAR-10, and TIMIT datasets. For MNIST and CIFAR-10 in particular, we analyze the spectrum of the EDJMs (see Figure 6), and find higher normalized singular values for the major components, which supports our intuition about putting the log-loss spectrum regularization on layer weights. On TIMIT, the spectrally regularized model also had higher accuracy than that trained with dropout. Interestingly, even though spectral regularization increased the EDJM score more than dropout, the resulting models had lower accuracy than dropout on MNIST and CIFAR-10.

6. Discussion

In this work, we introduce the Extended Data Jacobian Matrix, a novel tool to analyze deep neural networks of various structures conditioned on certain set of data. Especially for ReLU networks, we show an EDJM's equivalence to the underlying deep neural network. We find that the normalized singular values of the major components of EDJMs strongly correlate with the accuracy for networks of different structures. Based on such observation, we propose a novel regularization method, which manages to improve the network performance comparably to dropout. We believe the Extended Data Jacobian Matrix provides a tool for measuring the relative complexity of the functions learned by networks of different architecture on the manifold of interest defined by the data sample. For future work, statistics other than singular values, or the proposed score of EDJMs, can be intriguing, and may lead to new theoretical results about deep neural networks. We also plan to investigate EDJMs applied to recurrent networks.

References

Ba, Jimmy and Caruana, Rich. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pp. 2654–2662, 2014.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Hinton, GE, Vinyals, Oriol, and Dean, Jeff. Dark knowledge. *Presented as the keynote in BayLearn*, 2014.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29 (6):82–97, 2012.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Montufar, Guido F, Pascanu, Razvan, Cho, Kyunghyun, and Bengio, Yoshua. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pp. 2924–2932, 2014.

Neyshabur, Behnam, Salakhutdinov, Ruslan R, and Srebro, Nati. Path-sgd: Path-normalized optimization in deep neural networks. In Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2413–2421. Curran Associates, Inc., 2015.

Pascanu, Razvan, Montufar, Guido, and Bengio, Yoshua. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013.

Romero, Adriana, Ballas, Nicolas, Kahou, Samira Ebrahimi, Chassang, Antoine, Gatta, Carlo, and Bengio, Yoshua. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

Saxe, Andrew M, McClelland, James L, and Ganguli, Surya. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, van den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.