Insights regarding overfitting on noise in deep learning

Marthinus W. Theunissen[0000−0002−7456−7769], Marelie H. Davel[0000−0003−3103−5858], and Etienne Barnard[0000−0003−2202−2369]
{tiantheunissen, marelie.davel, etienne.barnard}@gmail.com

Multilingual Speech Technologies, North-West University, South Africa;
and CAIR, South Africa

Abstract. The understanding of generalization in machine learning is in a state of flux. This is partly due to the relatively recent revelation that deep learning models are able to completely memorize training data and still perform appropriately on out-of-sample data, thereby contradicting long-held intuitions about generalization. The phenomenon was brought to light and discussed in a seminal paper by Zhang et al. [24]. We expand upon this work by discussing local attributes of neural network training within the context of a relatively simple and generalizable framework. We describe how various types of noise can be compensated for within the proposed framework in order to allow the global deep learning model to generalize in spite of interpolating spurious function descriptors. Empirically, we support our postulates with experiments involving overparameterized multilayer perceptrons and controlled noise in the training data.
The main insights are that deep learning models are optimized for training data modularly, with different regions in the function space dedicated to fitting distinct kinds of sample information. Detrimental overfitting is largely prevented by the fact that different regions in the function space are used for prediction based on the similarity between new input data and that which has been optimized for.

Keywords: Deep learning · Machine learning · Learning theory · Generalization

1 Introduction

The advantages of deep learning models over some of their antecedents include their efficient optimization, scalability to high dimensional data, and performance on data that was not optimized for [7]. The latter is arguably the most important benefit. Machine learning, as a whole, has seen much progress in recent years, and deep neural networks (DNNs) have become a cornerstone in numerous important domains such as computer vision, natural language processing and bioinformatics. Somewhat ironically, the surge of application potential that deep learning has unlocked in industry has resulted in the development of theoretically principled guidelines lagging behind implementation-specific progress.

A particular example of one such open theoretical question is how to consolidate the observed ability of DNNs to generalize with classical notions of generalization in machine learning.

Before deep learning, a generally accepted principle with which to reason about generalization in machine learning was that it is linked to the complexity of the hypothesis space, and that the model's representational capacity should be kept small so as to prevent it from approximating unrealistically complex functions [22]. Such highly complex functions are not expected to be applicable to the task being performed, and are usually a result of overfitting the spurious correlations found in the finite sample of training examples. Many complexity metrics have been proposed and adapted in an attempt to consolidate this intuition, however, these metrics have consistently failed to robustly account for the generalization observed in deep learning models.

An influential paper by Zhang et al. [24] has demonstrated that: (1) DNNs can efficiently fit various types of noise and still generalize well, and (2) contemporary explicit regularization is not required to enable good generalization. These findings are in stark contradiction with complexity-based principles of generalization: deep learning models are shown to have representational capacities large enough to approximate extremely complex functions (potentially memorizing the entire data set) and still have very low out-of-sample test error.

In this paper we further investigate the effect of noise in training data with regards to generalization. Where [24] contributed towards the understanding of generalization by pointing out misconceptions with regards to model capacity and regularization, our contribution is to provide insight into a particular phenomenon that can (at least partially) account for the observed ability of a DNN to generalize in a very similar experimental framework. We define noise as any input-output relationship that is not predictable, or not conducive to the model fitting the true training data, or approximating the true data distribution. The following types of noise are investigated (detailed definitions are provided in appendix B):

1. **Label corruption:** For every affected sample, its training label is replaced with an alternative selected uniformly from all other possibilities.
2. **Gaussian input corruption:** For every affected sample, all its input features are replaced with Gaussian noise with unchanged mean and standard deviation.
3. **Structured input corruption:** For every affected sample, its input features are replaced by an alternative sample that is completely distinguishable from the true input but consistent per class. (For example, replacing selected images from one class with images of a completely different object from a different data set, but keeping the original class label.)

These types of noise have been chosen to encompass those investigated in [24], place more emphasis on varying the noise, and still maintain the analytically convenient per-sample implementation of the framework. It is possible to introduce other types of noise, such as stochastic corruptions in the feature representations at each layer within the architecture or directly impeding the optimization of the

model. However, such noise is very analogous to existing regularizing techniques (dropout, weight decay, node pruning etc.) and not aligned with the goals of this paper, namely, to shed light on how a DNN with as few as possible regularizing factors manages noisy data.

The empirical investigation will be limited to extremely overparameterized multilayer perceptron (MLP) architectures with ReLU activations, and 2 related classification data sets. These architectures are simple enough to allow for efficient analysis but function using the same principles as more complex architectures. The 2 classification tasks are MNIST [14] and FMNIST [23]. These data sets are on the low end of the spectrum with regard to difficulty, with FMNIST slighly more difficult than MNIST; both are widely used to investigate the theoretical properties of DNNs [9, 11, 20].

The following section (Section 2) discusses some recent, notable work that all have the goal of characterizing generalization in deep learning. In Section 3 a theoretical discussion is provided that defines a view of DNN training that is useful to interpret the results that follow. A detailed analysis of the ability of an MLP to respond to noisy data is provided and discussed in Section 4. Findings are summarized in the final section with a focus on implications relating to generalization.

2 Related work

Many attempts at conceptualizing the generalization ability of DNNs focus on the stability with which a model accurately predicts output values in the presence of varying input values. One popular approach is to analyse the geometry of the loss landscape at the optimum [8, 1, 12]. This amounts to investigating the overall loss value within a region of parameter space close to the optimal parameterization obtained by the training process. The intuition is that a sharper minimum or high curvature at the solution point indicates that the model will be sensitive to small perturbations in the input space. Logically, this will lead to poor generalization. A practical problem with this approach is that it suffers heavily from the curse of dimensionality. This means that it is difficult to obtain an unbiased and consistent perspective on the error surface in high dimensional parameter spaces, which is the case in virtually all practical DNNs. The error surface is typically mapped with dimensionality reductions, random searches, or heuristic searches [15]. A conceptual problem is that the loss value is easily manipulable by weight scaling. For example, Dinh et al. [3] have shown that a minimum can be made arbitrarily sharp/flat with no effect on generalization by exploiting simple symmetries in the weight scales of networks with rectified units.

Another effort at imposing stability in model predictions is to enforce sparsity in the parameterization. The hope is that with a sparsely connected set of trainable parameters, a reduced number of input parameters will affect the prediction accuracy. Like complexity metrics, this idea is borrowed from statistical learning theory [17] and with regards to deep learning, it has seen more

success in terms of improving the computational cost [5, 16] and interpretability of DNNs than in improving generalization.

From a representational point of view, some have argued that the functions that overparameterized DNNs approximate are inherently insensitive to input perturbations at the optimums to which they converge [20, 21, 18, 19]. These investigations place a large emphasis on the design choices (depth, width, activation functions, etc.) and are typically exploratory by nature.

A new approach proposed by [4] and [10] investigates DNN generalization by means of "margin distributions", a measure of how far samples tend to be from the decision boundary. These types of metrics have been successfully used to indicate generalization ability in linear models such as support vector machines (SVMs); however, determining the decision boundary in a DNN is not as simple. Therefore, they used the first-order Taylor approximation of the distance to the decision boundary between the ground truth class and the second highest ranking class. They were able to use a linear model, trained on the margin distributions of numerous DNNs, to predict the generalization error of several out-of-sample DNNs. This suggests that DNN generalization is strongly linked to the type of representation the network uses to represent sample information throughout its layers.

3 Hidden layers as general feature space converters

Each hidden layer in an MLP is known to act as a learned representation, converting the feature space from one representation to another. At the same time, individual nodes act as local decision makers and respond to very specific subsets of the input space, as discussed below.

3.1 Per-layer feature representations

In a typical MLP training scenario, several hidden layers are stacked in series. The output layer is the only one which is directly used to perform the global task. The role that the hidden layers perform is one of enabling the approximation of the necessary non-linear function through the use of the non-linearities produced by their activation functions. In this sense, all hidden layers can be thought of as general feature space converters, where the behaviour of one dimension in one feature space is determined by a weighted contribution of all the dimensions in the preceding feature space. Refer to Fig. 1 for a visual illustration of this viewpoint.

3.2 Per node decision making

If every hidden layer produces a feature space, then every node determines a single dimension in this feature space. Some insights can be gained by theoretically describing the behaviour of a node in terms of activation patterns in the preceding layer. Let $\mathbf{a}_l$ be an activation vector at a layer l as a response to an

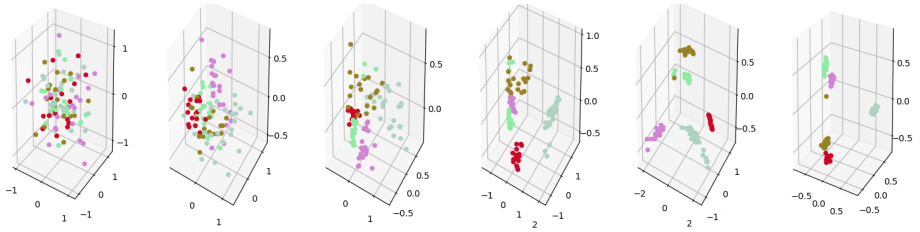


Fig. 1: An illustration of feature spaces [from left to right: input; 4 x hidden layers; output layer] in a trained MLP. The model was trained to perform a 5-class classification task of 100 randomly generated 50 dimensional input vectors. Note that principle component analysis is used to reduce the dimensionality of the actual feature spaces to 3 for this visual depiction.

input sample $\mathbf{x}$. If $\mathbf{W}_l$ is the weight matrix connecting l and the previous layer $l-1$ then:

$$\mathbf{a}_l = f_a\left(\mathbf{W}_l \cdot \mathbf{a}_{l-1}\right) \tag{1}$$

where f_a is some element-wise non-linear activation function. For every node i in l the activation value is then:

$$a_l^i = f_a\left(\mathbf{w}_l^i \cdot \mathbf{a}_{l-1}\right) \tag{2}$$

where $\mathbf{w}_l^i$ is the row in matrix $\mathbf{W}_l$ connecting layer $l-1$ and node i. This can be rewritten as:

$$a_l^i = f_a\left(\|\mathbf{w}_l^i\|\|\mathbf{a}_{l-1}\| cos\theta\right) \tag{3}$$

with θ specifying the angle between the weight vector and the activation vector. The pre-activation node value is determined by the product of the norm of the activation vector (in the previous layer) and the norm of the relevant weight vector (in the current layer) scaled by the cosine similarity of the two vectors. As a result, if the activation function is a rectified linear unit (ReLU) [6] and a bias is not used, the angle between the activation vector and the weight vector has to be $\in (-90°, 90°)$ for the sample information to be propagated by the node. In other words, the node is activated for samples that produce an activation pattern in the preceding layer with a cosine similarity larger than 0 in terms of the weight vector. This criterion holds regardless of the activation or weight strengths. (When a bias is used, the threshold angles are different, but the concept remains the same.)

3.3 Sample sets

When training a ReLU-activated network, the corresponding weight vector is only updated to optimize the global error in terms of *those specific samples for which a node is active*, referred to as the "sample set" of that node. This is because weights are updated according to the gradients which can only propagate

back through the network to the weight vector if the corresponding node is active. In this sense, the node weight vector acts as a hyperplane in the feature space of the preceding layer. This hyperplane corresponds to the points where the weight vector is equal to zero (or the bias value if one is present). Samples which are located on one side of the hyperplane are prevented from having an effect on the weight vector values, and samples on the other side are used to update the weights, thereby dictating the behaviour of one dimension in the following feature space. The actual weight updates are affected by the representations of sample information in all the layers following the current one. This phenomenon has been investigated and discussed in [2], where several types of per-layer classifiers were constructed from the nodes of trained MLPs. These classifiers were shown to perform at levels comparable to the global network from which they were created.

To summarize this theoretical view point of ReLU-activated MLP training:

1. Hidden layers represent sample information in a feature space unique to every layer.
2. The representations of sample information are created on a per dimension basis by the weight vectors and activation functions linking the nodes to the preceding layer.
3. A node is only active for samples with a directional similarity in the previous feature space.
4. These sets of samples are used to update the weight vectors during training and (by extension) the representation used by the following feature space.

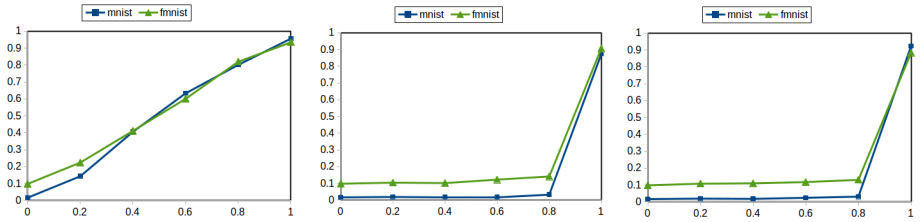
4 Noise in the data

4.1 Model performance

In order to investigate the sample sets and activation/weight vector interactions, several MLPs, containing 10 hidden layers of 512 nodes each, are trained on varying levels of noise. A standard experimental setup is used, as described in appendix A. Fig. 2 shows the resulting performance of the different models, when tested on uncorrupted test data. All models were able to easily fit the noisy training data, corroborating the findings of [24].

Notice that, when analyzing label corruption, there is a near linear inverse correlation between the amount of label noise and the model's ability to generalize to unseen data. This suggests that either:

1. the models are memorizing sample-specific input-output relationships and a certain portion of the unseen data is similar enough to a corresponding portion of uncorrupted training samples to facilitate appropriate generalization; or
2. the global approximated function is somehow compartmentalised to contain fundamental rules about the task in some regions and ad hoc rules with which to correctly classify the corrupted samples in other regions.



(a) label corruption (b) gaussian input corruption (c) structured input corruption

Fig. 2: The generalization gap for models trained on MNIST (blue) and FMNIST (green) at varying levels of three types of noise. The horizontal axis represents the probability of any given training sample having been corrupted for the relevant model. All models are overparameterized and have perfect, or close to perfect, performance on the training data.

Observe from the results of the two input corruptions that noise in the input data has an *almost negligible* effect on generalization up to the point where there is an insufficient amount of true data in the set with which to learn. This threshold is expected to change with more difficult classification tasks (more class variance and overlap), data sets containing fewer samples in total, and models with less parameter flexibility.

The fact that input noise does not result in a linear reduction in generalization ability still supports both the previous propositions. If the first postulate is true, then the samples with corrupted input data are memorized, but no samples in the evaluation set are similar enough to them to incur additional generalization error. If the second postulate is true, then the regions in the approximated function that were determined by the corrupted input samples are simply never used for classifying the uncorrupted evaluation set.

It is also worth noting that the Gaussian and structured input corruptions have very similar influences on generalization. The models are therefore able to generalize in the presence of input noise regardless of its predictability.

4.2 Cosine similarities

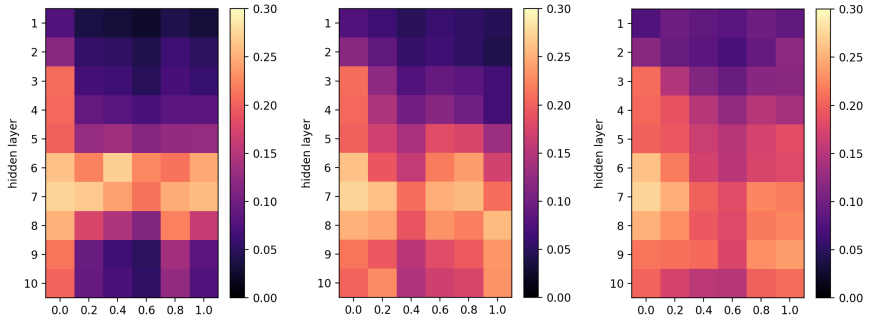
The cosine similarity ($\cos\theta$ as also used in Eq. 3) can be used as an estimate of how much the representation of a sample in a preceding layer is directionally similar to that of the set of samples for which a node tends to be active. By measuring the average cosine similarity of samples in a node's sample set with regards to the determinitive weight vector (w_l^i in Eq. 3) and averaging over nodes in a layer, it is possible to identify layers where samples tend to be grouped together convincingly. That is, the samples are closely related (in the preceding feature space) and the resulting activation values tend to be large.

Using Eq. 3, the mean cosine similarity per-layer l (over all weight and active sample pairs at every node) can be calculated as:

$$\mu_{\text{cosine}}(l) = \frac{1}{|N_l|} \sum_{i \in N_l} \left(\frac{1}{|A_i|} \sum_{a \in A_i} \frac{a_l^i}{\|\mathbf{w}_l^i\| \|\mathbf{a}_{l-1}\|} \right) \tag{4}$$

where N_l is a set of all the nodes in layer l and A_i is a set of all positive activation values at node i.

Fig. 3 shows this metric for models trained on various amounts of noise. It can be observed that noise in either the input or labeling data results in the depth at which high mean cosine similarities are obtained being deeper in the noise-corrupted networks compared to the baseline models. Additionally, take note that the cosine similarities for the structured input noise are more spread out across layers than the other two types of noise, which is more consistent with the baseline model. Lastly, it is clear from the results of the models containing noise in the labeling data that a "convincing" representation of training data is obtained at around layer 6. Very small cosine similarities are observed in the earlier layers of all models under any noise conditions.



(a) label corruption (b) gaussian input corruption (c) structured input corruption

Fig. 3: Mean cosine similarity per-layer at varying levels of three types of noise. The horizontal axis represents the probability of any given training sample having been corrupted for the relevant model. Results are measured on the training samples of the MNIST data set. Equivalent results for FMNIST are included in Appendix C.

4.3 Sample set corruption composition

The noise in the data sets is introduced probabilistically on a per sample basis (see Alg. 1 to 3). This provides a convenient way to investigate the composition

of sample sets with regards to noise. Fig. 4 shows how sample sets can consist of different ratios of true and corrupted sample information.

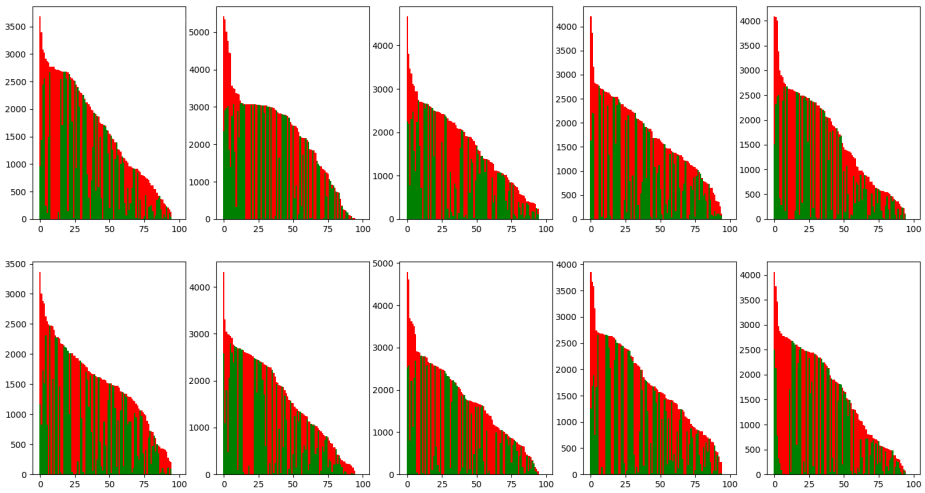


Fig. 4: Per class sample set corruption ratios for the first hidden layer of a 3x100 MLP fitting MNIST training examples, including structured input corruptions at a probability of 0.5. The nodes have been arranged in descending order of sample set size. The true and corrupted portions of the sample sets are presented in green and red, respectively.

We define the *polarization* of a node for a class as the amount the sample set of a node i favours either the corrupted or uncorrupted samples of a class c, respectively. This is defined as follows:

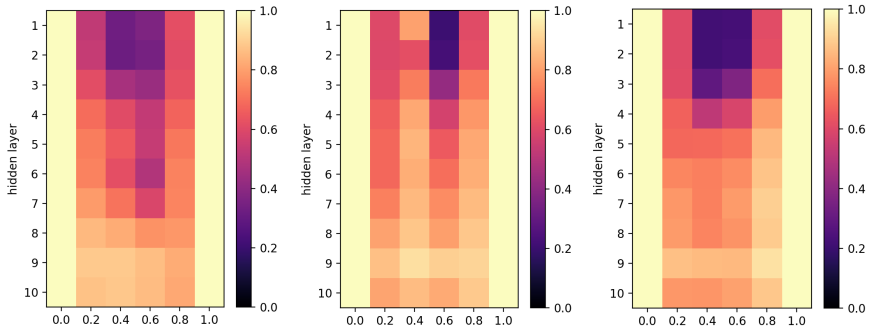
$$\text{polarization}(c, i) = 2 \left| \frac{1}{2} - \frac{|\widetilde{A_i^c}|}{|A_i^c|} \right| \tag{5}$$

where A_i^c is a set of all positive activation values at node i in response to samples from class c, and $\widetilde{A_i^c}$ is a corresponding set limited to corrupted samples. By averaging over nodes and classes, a per-layer mean polarization values can be obtained with the following equation:

$$\mu_{\text{polarization}}(l) = \frac{1}{|K||N_l|} \sum_{c \in K, i \in N_l} \text{polarization}(c, i) \tag{6}$$

where K is a set of all classes. Dead nodes (that are not activated for any sample) are omitted when performing the averaging operations. A dead node does not contribute to the global function and merely reduces the dimensionality of the feature space in the corresponding layer by 1.

The polarization metric indicates how much the sample sets formed within a layer are polarized between true class information and corrupted class information, for any given class. The relevant polarization values are provided in Fig. 5. The main observation is that sample sets tend to be highly in favour of true **or** corrupted sample information, and this is especially true in the later layers of any given model. The label and structured input corruption produces lower polarization earlier in the model, but this is to be expected seeing as the input data has strong coherence based on correlations in class-related input structures. These findings support the second postulate in section 4.1. It appears that sub-regions in the function space are dedicated to processing different kinds of training data.



(a) label corruption (b) gaussian input corruption (c) structured input corruption

Fig. 5: Mean per-layer corruption polarization at varying levels of three types of noise. The horizontal axis represents the probability of any given training sample having been corrupted for the relevant model. Results are measured on the training samples of the MNIST data set. Equivalent results for FMNIST are included in Appendix C.

4.4 Discussion

In this section we have shown that several overparameterized DNNs with no explicit regularization are able to generalize well with evident spurious input-output relationships present in the training data. We have used empirically evaluated metrics to show that, in the presence of noise, well-separated per node sample sets are generated later in the network compared to baseline cases with no noise. Additionally, these well-separated sets of samples are highly polarized between samples containing true task information and samples without.

If we accept the viewpoint that nodes in hidden layers act as collaborating feature differentiators (separating samples based on feature criteria that are unique to each node) to generate informative feature spaces, then each layer

also acts as a mixture model fitting samples based on their representation in the preceding layer. A key insight is that each model component is not fitted to all samples in the data set. Model components (referring to a node and its corresponding weight vector) are optimized on a specific subset of the population as determined by the activation patterns in the preceding layer. And, as we have observed, these subpopulations can and tend to be composed of true task information or false task information.

In this sense, some model components of the network are dedicated to correctly classifying uncorrupted samples, and others are dedicated to corrupted samples. To generalize this observation to training scenarios without explicit data corruption, it can be observed that in most data sets samples from a specific class have varied representations. Without defining some representations as noise, they are still processed in the same way the structured input corruption data is processed in this paper, hence the strong coherence between the baseline models and those containing structured input noise. This is also why it is possible to perform some types of multitask learning. One example would be combining MNIST and FMNIST. In this scenario the training set will contain 120 000 examples with consistent training labels, but two distinct representations in the input space. For example, class 6 will be correctly assigned to the written number 6 and a shirt.

To summarise, DNNs do overfit on noise, albeit benignly. The vast representational capacity and non-linearities enable subcomponents of the network to be dedicated to processing subpopulations of the training data. When out-of-sample data is to be processed, the regions most similar to the unseen data is used to make predictions, thereby preventing the model components fitted to noise from affecting generalization.

5 Conclusion

We have investigated the phenomenon that DNNs are able to generalize in spite of perfectly fitting noisy training data. An interesting mechanism that is intrinsic to non-linear function approximating in deep MLPs has been described and supporting empirical analyses have been provided. The findings in this paper suggest that good generalization in large DNNs, in spite of extreme noise in the training data, is a result of the modular way training samples are fitted during optimization. Future work will attempt to construct a formal framework with which to characterize the collaborating sub-components and, based on this, possibly produce some theoretically grounded predictors of generalization.

6 Acknowledgements

This work was partially supported by the National Research Foundation (NRF, Grant Number 109243). Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and the NRF does not accept any liability in this regard.

References

1. Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J.T., Sagun, L., Zecchina, R.: Entropy-SGD: Biasing gradient descent into wide valleys. ArXiv **abs/1611.01838** (2016)
2. Davel, M.H., Theunissen, M.W., Pretorius, A.M., Barnard, E.: DNNs as layers of cooperating classifiers. In: AAAI 2020 (submitted for publication)
3. Dinh, L., Pascanu, R., Bengio, S., Bengio, Y.: Sharp minima can generalize for deep nets. arXiv preprint **arXiv:1703.04933v2** (2017)
4. Elsayed, G.F., Krishnan, D., Mobahi, H., Regan, K., Bengio, S.: Large margin deep networks for classification. In: NeurIPS (2018)
5. Gale, T., Elsen, E., Hooker, S.: The state of sparsity in deep neural networks. ArXiv **abs/1902.09574** (2019)
6. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: AISTATS (2011)
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
8. Hochreiter, S., Schmidhuber, J.: Flat minima. Neural Computation **9**, 1–42 (1997)
9. Jastrzebski, S., Kenton, Z., Ballas, N., Fischer, A., Bengio, Y., Storkey, A.J.: On the relation between the sharpest directions of dnn loss and the sgd step length. In: ICLR (2018)
10. Jiang, Y., Krishnan, D., Mobahi, H., Bengio, S.: Predicting the generalization gap in deep networks with margin distributions. arxiv preprint (In ICLR 2019) **arXiv:1810.00113v2** (2019)
11. Kawaguchi, K., Kaelbling, L.P., Bengio, Y.: Generalization in deep learning. ArXiv **abs/1710.05468** (2017)
12. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: Generalization gap and sharp minima. ArXiv **abs/1609.04836** (2016)
13. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint (In ICLR 2014) **arXiv:1412.6980** (2014)
14. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
15. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: NeurIPS (2017)
16. Loroch, D.M., Pfreundt, F.J., Wehn, N., Keuper, J.: Sparsity in deep neural networks - an empirical investigation with tensorquant. In: DMLE/IOTSTREAMING@PKDD/ECML (2018)
17. Maurer, A., Pontil, M.: Structured sparsity and generalization. Journal of Machine Learning Research **13** (2011)
18. Montúfar, G., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. ArXiv **abs/1402.1869** (2014)
19. Neal, B., Mittal, S., Baratin, A., Tantia, V., Scicluna, M., Lacoste-Julien, S., Mitliagkas, I.: A modern take on the bias-variance tradeoff in neural networks. ArXiv **abs/1810.08591** (2018)
20. Novak, R., Bahri, Y., Abolafia, D.A., Pennington, J., Sohl-Dickstein, J.: Sensitivity and generalization in neural networks: an empirical study. In: International Conference on Learning Representations (ICLR) (2018)
21. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., Sohl-Dickstein, J.: On the expressive power of deep neural networks. In: Proceedings of the 34th International Conference on Machine Learning. pp. 2847–2854 (2017)

22. Vapnik, V.N.: An overview of statistical learning theory. IEEE Transactions on Neural Networks and Learning Systems (1999)
23. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747v2 (2017)
24. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. arXiv preprint (In ICLR 2017) **arXiv:1611.03530** (2016)

A Experimental setup

The classification data sets that are used for empirical investigations are MNIST [14] and FMNIST[23]. These data sets are drop-in replacements for each other, meaning that they contain the same number of input dimensions, classes, and examples. Namely, 28x28 (784 flattened), 10, and 70 000, respectively. In all training scenarios a random split of 55 000 training examples, 5 000 validation examples, and 10 000 evaluation examples are used. The generalization gap is calculated by subtracting the evaluation accuracy from the training accuracy.

All models are trained for 400 epochs with randomly selected batches of 128 examples. This amounts to 171 600 parameter updates in total. In order to ensure that the training data is completely fitted, this is repeated for at least 2 random uniform parameter initialization and the model that best fits the training data is selected to be analysed. A fixed MLP architecture containing 10 hidden layers of 512 nodes each is used, with a single bias node at the first layer. ReLU activation functions are used at all hidden layers. The popular Adam [13] optimizer is used with a simple mean squared error (MSE) loss function at the output. No output activation function is employed and no additional regularizing techniques are implemented. That includes batch normalization, weight decay, drop out, data augmentation, and early stopping.

B Algorithms for adding noise

The first type of noise is identical to the "Partially corrupted labels" in [24] except for the selection of alternative labels. Instead of selecting a random label uniformly, we select a random alternative (not including the true label) uniformly. This results in a data set that is corrupted an amount closer to the one represented by the probability value P that determines corruption levels. See Algorithm 1. The second type of noise is similar to the "Gaussian" in [24], with the difference being that instead of replacing input data with Gaussian noise selected from a variable with identical mean and variance to the data set, we determine the mean and variance of the Gaussian in terms of the specific sample being corrupted. See Algorithm 2. The third type of noise replaces input data with alternatives that are completely different to any in the true data set but still structured in a way that is predictable. This is accomplished by rotating the sample 90° counter-clockwise about the center, followed by an inversion of the feature values. Inversion refers to subtracting the feature value from the maximum feature value in the sample. See Algorithm 3.

Algorithm 1: Label corruption

Input: A train set of labelled examples ($\mathbf{X}^{(train)}$, $\mathbf{Y}^{(train)}$), a set of possible labels $\mathbf{C}$, and a probability value P

Output: A train set of corrupted examples ($\mathbf{X}^{(train)}$, $\hat{\mathbf{Y}}^{(train)}$)

for y *in* $\mathbf{Y}^{(train)}$ **do**
  if $\mathcal{B}(1, P)$ **then**
    $\hat{y} \sim \mathcal{U}[\mathbf{C} \backslash \{y\}]$
  else
    $\hat{y} \leftarrow y$

Algorithm 2: Gaussian input corruption

Input: A train set of labelled examples ($\mathbf{X}^{(train)}$, $\mathbf{Y}^{(train)}$), a set of possible labels $\mathbf{C}$, and a probability value P

Output: A train set of corrupted examples ($\hat{\mathbf{X}}^{(train)}$, $\mathbf{Y}^{(train)}$)

for $\mathbf{x}$ *in* $\mathbf{X}^{(train)}$ **do**
  if $\mathcal{B}(1, P)$ **then**
    $\hat{\mathbf{x}} \leftarrow \mathbf{g}$
    where $\mathbf{g}$ is a vector sampled from $\mathcal{N}[\mu_{\mathbf{x}}, \sigma_{\mathbf{x}}]$
  else
    $\hat{\mathbf{x}} \leftarrow \mathbf{x}$

Algorithm 3: Structured input corruption

Input: A train set of labelled examples ($\mathbf{X}^{(train)}$, $\mathbf{Y}^{(train)}$), a set of possible labels $\mathbf{C}$, and a probability value P

Output: A train set of corrupted examples ($\hat{\mathbf{X}}^{(train)}$, $\mathbf{Y}^{(train)}$)

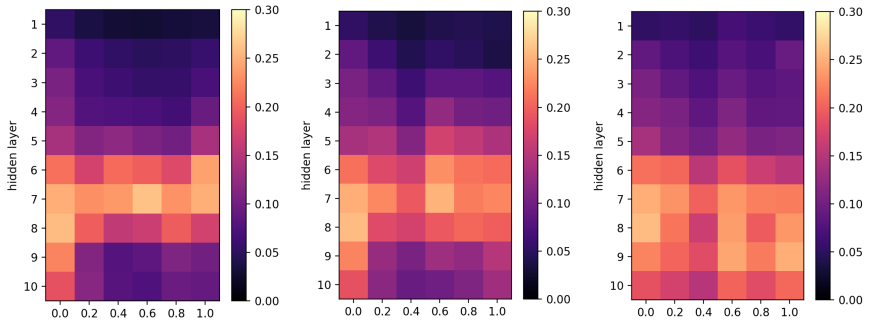
for $\mathbf{x}$ *in* $\mathbf{X}^{(train)}$ **do**
  if $\mathcal{B}(1, P)$ **then**
    $\hat{\mathbf{x}} \leftarrow$ **invert**(**rotate**($\mathbf{x}$))
  else
    $\hat{\mathbf{x}} \leftarrow \mathbf{x}$

rotate is a 90° rotation counter clockwise about the origin
invert is an inversion of all values in the vector

C Additional results

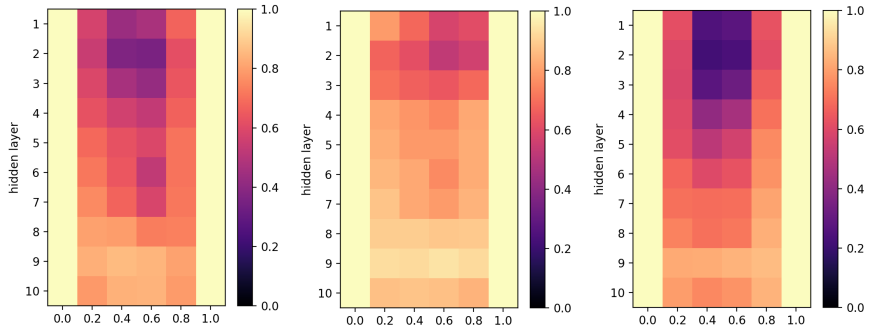
The mean cosine similarities and mean polarization values for analyses conducted on the FMNIST data set are provided in Fig. 6 and 7, respectively. Notice that most of the same observations can be made when compared to the MNIST results in section 4.2 and 4.3. It is, however, worth noting that for a classification task

with more overlap in the input space such as FMNIST, the well-separated sample sets are generated at even later layers and the polarization is higher overall.



(a) label corruption (b) gaussian input corruption (c) structured input corruption

Fig. 6: Mean cosine similarity per-layer at varying levels of three types of noise. The horizontal axis represents the probability of any given training sample having been corrupted for the relevant model. Results are measured on the training samples of the FMNIST data set.



(a) label corruption (b) gaussian input corruption (c) structured input corruption

Fig. 7: Mean per-layer corruption polarization at varying levels of three types of noise. The horizontal axis represents the probability of any given training sample having been corrupted for the relevant model. Results are measured on the training samples of the FMNIST data set.