MEASURING THE INTRINSIC DIMENSION OF OBJECTIVE LANDSCAPES

Anonymous authors
Paper under double-blind review

ABSTRACT

Many recently trained neural networks employ a large number of parameters to achieve good performance. One may intuitively use the number of parameters required as a rough gauge of the difficulty of a problem. How many parameters are really needed? In this paper we attempt to answer this question through investigating the objective landspaces constructed by the datasets and neural networks. Instead of training networks directly in the extrinsic parameter space, we propose training in a smaller, randomly oriented subspace. The solutions first appear, by gradually increasing the dimension of this subspace to some number, as which we define the *intrinsic dimension* of the objective landspace. A few suggestive conclusions result. Many problems have smaller intrinsic dimensions than one might suspect, and the intrinsic dimension for a given dataset varies little across a family of models with vastly different sizes. Intrinsic dimension allows some quantitative problem comparison across supervised and reinforcement learning where we conclude, for example, that solving the inverted pendulum problem is 100 times easier than classifying digits from MNIST. In addition to providing new cartography of the objective landscapes wandered by parameterized models, the results encompass a simple method for constructively obtaining an upper bound on the minimum description length of a solution, leading to very compressible networks in some cases.

1 INTRODUCTION

In this paper we investigate and define the *intrinsic dimension* of neural network optimization problems. To introduce this concept, we first briefly review neural network training from an optimization perspective. Modeling a given dataset using neural networks to entails two main steps. First, a network designer chooses a loss function and a network architecture for a given dataset. Second, the architecture is then initialized by populating its weights with random values drawn from some distribution, and the network is trained by adjusting its weights to produce a loss for the dataset that is as low as possible.

Revisting this procedure, we can think of the training as traversing some path along an *objective landscape*. The landscape in its entirety is completely determined in the first step, once the *dataset* and *network architecture* are specified. It is instantiated and frozen; all subsequent optimization in the second step, including parameter initialization, forward/backward propagation, and gradient-based updates via some optimizers, are just details of how one explores this fixed space.

Consider a network parameterized by D weights. We can picture its associated objective landscape as a set of "hills and valleys" in D dimensions, where each point in $\mathbb{R}^D$ corresponds to a value of the loss (*i.e.,* the elevation of the landscape). When $D = 2$, the map from two coordinates to one can be easily imagined, and intuitively analogized to similar hills in our three-dimensional world. However, in higher dimensions, our intuitions may not be so faithful, as extrapolating from low-dimensional intuitions to higher-dimensions can lead to unreliable conclusions. Notwithstanding the difficulty of understanding the high-dimensional landscapes, it is the lot of neural network researchers to spend their efforts leading (or following?) networks over these multi-dimensional surfaces. Therefore, any interpreted geography of these landscapes is valuable.

Several papers have shed valuable light on this landscape, particularly by pointing out flaws in our extrapolation from low-dimensional reasoning. Dauphin et al. (2014) showed that, in contrast

to conventional thinking about getting stuck in local optima (as one might be stuck in a valley in our familiar $D = 2$), local critical points in high dimension are almost never valleys but are instead saddlepoints: structures which are "valleys" along a multitude of dimensions with "exits" in a multitude of other dimensions. The striking conclusion is that one has less to fear becoming hemmed in on all sides by higher loss but more to fear being waylaid nearly indefinitely by nearly flat regions. Goodfellow et al. (2015) showed another property that paths directly from the initial point to the final point of optimization are often monotonically decreasing. Though dimension is high, the space is in some sense simpler than we thought: rather than winding around hills and through long twisting corridors, the walk could just as well have taken a straight line without encountering any obstacles, if only the direction of the line could have been determined at the outset.

In this paper we seek further understanding of the structure of the objective landscape by restricting training to random slices through it, allowing optimization to proceed in randomly generated subspaces of the full parameter space. Whereas the standard neural network training involves computing a gradient and taking a step in the full parameter space ($\mathbb{R}^D$ above), we instead choose a random d-dimensional subspace of $\mathbb{R}^D$, where generally $d < D$, and optimize directly in this subspace. By performing experiments with gradually larger values of d, we can find the subspace dimension at which solutions first appear, which we call the measured *intrinsic dimension* of a particular landscape. Examining intrinsic dimensions across a variety of problems leads to a few new intuitions about the optimization problems that arise from neural network models. We begin by defining more precisely the notion of intrinsic dimension.

2 Defining and Approximating Intrinsic Dimension

We introduce the intrinsic dimension of an objective landscape with an illustrative toy problem. Let $\theta^{(D)} \in \mathbb{R}^D$ be a parameter vector in the parameter space of dimension D, $\theta_0^{(D)}$ a randomly chosen initial parameter vector, and $\theta_*^{(D)}$ the final parameter vector arrived at via optimization.

Consider a toy optimization problem where $D = 1000$, $\theta_0^{(D)}$ is drawn from a Gaussian distribution, and $\theta^{(D)}$ optimized to minimize the objective $J(\theta) = \sum_{r=1}^{10} \|1 - \sum_{i=100r-99}^{100r} \theta_i\|^2$, where θ_i is the ith element of $\theta^{(D)}$. We may optimize in $\mathbb{R}^D$ to find a $\theta_*^{(D)}$ that solves the problem with zero cost.

Solutions to this problem are highly redundant. One can also analytically find that the manifold of solutions is a 990 dimensional hyperplane: from any point that has zero cost, there are 990 orthogonal directions one can move and remain at zero cost. Denoting s as the dimensionality of the solution set, we define the intrinsic dimensionality d_{int} of a solution as the codimension of the solution set inside of $\mathbb{R}^D$:

$$D = d_{\text{int}} + s \tag{1}$$

Here the intrinsic dimension d_{int} is 10 (1000 = 10 + 990), with 10 corresponding intuitively to the number of constraints placed on the parameter vector.

2.1 Measuring Intrinsic Dimension via Random Subspace Training

The above example has a simple enough form that we obtained $d_{\text{int}} = 10$ by calculation. But in general we desire a method to measure or approximate d_{int} for more complicated problems, including problems with data-dependent objective functions, e.g. neural network training. Random subspace optimization provides such a method.

Standard optimization, which we will refer to hereafter as the *direct* method of training, entails evaluating the gradient of a loss with respect to $\theta^{(D)}$ and taking steps directly in the space of $\theta^{(D)}$. To train in a random subspace, we instead define $\theta^{(D)}$ in the following way:

$$\theta^{(D)} = \theta_0^{(D)} + P\theta^{(d)} \tag{2}$$

where P is a randomly generated $D \times d$ projection matrix[1] and $\theta^{(d)}$ is a parameter vector in a generally smaller space $\mathbb{R}^d$. $\theta_0^{(D)}$ and P are randomly generated and frozen (not trained), so the

[1]This projection matrix can take a variety of forms each with different computational considerations. In later sections we consider dense, sparse, and implicit P matrices.

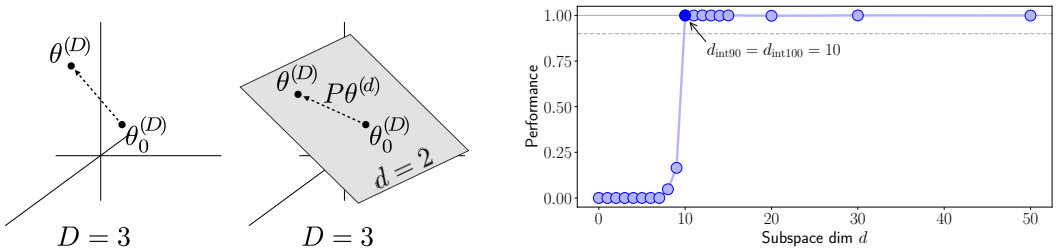


Figure 1: **(left)** Illustration of parameter vectors for direct optimization in the $D = 3$ case. **(middle)** Illustration of parameter vectors and a possible random subspace for the $D = 3, d = 2$ case. **(right)** Plot of performance *vs.* subspace dimension for the toy example of toy example of Sec. 2. The problem becomes both 90% solvable and 100% solvable at random subspace dimension 10, so d_{int90} and d_{int100} are 10.

system has only d degrees of freedom. We initialize $\theta^{(d)}$ to a vector of all zeros, so initially $\theta^{(D)} = \theta_0^{(D)}$, a setting that serves two related purposes: First, it puts the subspace used for training and the solution set in *general position* with respect to each other. Intuitively, this avoids cases where both the solution set and the random subspace contain structure oriented around the origin, which could bias toward non-intersection even when $d + s > D$. Second, in the case of neural network training, it allows the network to benefit from good initialization schemes (Glorot & Bengio, 2010; He et al., 2015): beginning in a well-conditioned region of the parameter space such that gradient descent via commonly used optimizers tends to work well. It allows decoupling of the initialization of the network from the random subspace optimization process itself.

Training proceeds by computing gradients with respect to $\theta^{(d)}$ and taking steps in that space. Columns of P are normalized to unit length, so steps of unit length in $\theta^{(d)}$ chart out unit length motions of $\theta^{(D)}$. Columns of P may also be orthogonalized if desired, but in our experiments we relied simply on the approximate orthogonality of high dimensional random vectors. By this construction P forms an approximately orthonormal basis for a randomly oriented d dimensional subspace of $\mathbb{R}^D$, with the origin of the new coordinate system at $\theta_0^{(D)}$. Fig. 1 shows an illustration of the related vectors.

Consider a few properties of this training approach. If $d = D$ and P is a large identity matrix, we recover exactly the direct optimization problem. If $d = D$ but P is instead a random orthonormal basis for all of $\mathbb{R}^D$ (just a random rotation matrix), we recover a rotated version of the direct problem. Note that for some "rotation-invariant" optimizers, such as SGD and SGD with momentum, rotating the basis will not change the steps taken, but for optimizers with axis-aligned assumptions, such as RMSProp (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2014), the path taken through $\theta^{(D)}$ space by an optimizer will depend on the rotation chosen. Finally, in the general case where $d < D$ and solutions exist in D, solutions will *almost surely* (with probability 1) not be found if d is less than the codimension of the solution. On the other hand, when $d \geq D - s$, if the solution set is a hyperplane, the solution will almost surely intersect the subspace, but for solution sets of arbitrary topology, intersection is not guaranteed. Nonetheless, by iteratively increasing d, re-running optimization, and checking for solutions, we obtain one estimate of d_{int}. We try this sweep of d for our toy problem, measuring the positive performance instead of loss[2] in keeping with the convention described in the next section. As expected, the solutions are first found at $d = 10$ (see Fig. 1, right).

2.2 Details and Conventions

In the rest of this paper, we measure intrinsic dimensions for particular neural network problems and draw conclusions about the associated objective landscapes and solution sets. Because modeling real data is more complex than the above toy example, and losses are generally never exactly zero, we first choose a heuristic for classifying points on the objective landscape as solutions *vs.* non-

[2]For this toy problem we define $\text{performance} = \exp(-\text{loss})$, bounding performance between 0 and 1, with 1 being a perfect solution.

solutions. The heuristic we choose is to threshold network performance at some level relative to a baseline model, where generally we take as baseline the best directly trained model. In supervised classification settings, validation accuracy is used as the measure of performance, and in reinforcement learning scenarios, total reward (shifted up or down such that the minimum reward is 0) is used. Accuracy and reward are preferred to loss to ensure results are grounded to real-world performance and to allow comparison across models with differing scales of loss and different amounts of regularization included in the loss.

We define d_{int100} as the intrinsic dimension of the "100%" solution: those solutions whose performance is statistically indistinguishable from baseline solutions. However, when we attempted to measure d_{int100}, we observed it to vary widely and for a few confounding reasons: d_{int100} can be very high — nearly as high as D — when the task requires matching a very well-tuned baseline model, but can drop significantly when the regularization effect of restricting parameters to a subspace boosts performance by tiny amounts. While these are interesting effects, we primarily set out to measure the basic difficulty of problems and the degrees of freedom needed to solve (or approximately solve) them rather than these effects.

Thus, we found it more practical and useful to define and measure d_{int90} as the intrinsic dimension of the "90%" solution: those with performance at least 90% of the baseline. We chose 90% after looking at a number of dimension vs. performance plots (e.g. Fig. **??**) as a reasonable trade off between wanting to guarantee solutions are as good as possible, but also wanting measured d_{int} values to be robust to small noise in measured performance. If too high a threshold is used, then the dimension at which performance crosses the threshold changes a lot for only tiny changes in accuracy, and we always observe tiny changes in accuracy due to training noise. If a somewhat higher (or lower) threshold were chosen, we expect most of conclusions in the rest of the paper to remain qualitatively unchanged, albeit with d_{int} values scaled up (or down) by some factor and more noisily (or less noisily) measured. In the future researchers may find it useful to measure d_{int} using higher or lower thresholds.

The 90% threshold was empirically chosen based on the observation over a variety of dimension *vs.* performance plots: it is a reasonable trade off between maitaining good solutions (and a robust measurement) for individual problems and crafting a sensible measurement across multiple problems. When the threshold is high, the dimension of crossing the threshold changes significantly for only tiny changes of accuracy, which is usually due to training noise. Further, Giryes et al. (2016) showed that the main structure of the data manifold is prevserved via random weights of neural networks, and the role of training is to refine and encode the details of decision boundary. In our case, d_{int90} is designed to reflect the macro structures of the objective landspace. Therefore, if too high a threshold was used, we expect most of results in the rest of the paper to remain qualitatively unchanged, with d saturated in measuring the training noise and landspace details.

3 Results and Discussion

3.1 MNIST

We begin by analyzing a fully connected (FC) classifier trained on MNIST. We choose a 784–200–200–10 network, containing two hidden layers of width 200 and giving a total number of parameters $D = 199{,}210$. Performance *vs.* subspace dimension d is shown in Fig. 2 (left). By checking the subspace dimension at which performance crosses the 90% mark, we measure this network's intrinsic dimension d_{int90} at about 750.

Some networks are very compressible. Perhaps the most salient initial conclusion is that 750 is quite low. At that subspace dimension, only 750 degrees of freedom (0.4%) are being used and 198,460 (99.6%) unused to obtain 90% of the performance of the direct baseline model. A compelling corollary of this result is a simple, new way of creating and training compressed networks, particularly networks for applications in which the absolute best performance is not critical. To store this network, one need only store a tuple of three items: (i) the random seed to generate the frozen $\theta_0^{(D)}$, (ii) the random seed to generate P and (iii) the 750 floating point numbers in $\theta_*^{(d)}$. It leads to the compression (assuming 32-bit floats) from 793kB to only 3.2kB or 0.4% of the full parameter size. Such compression could be very useful for scenarios where storage or bandwidth are limited, including neural networks in app downloads or on web pages.

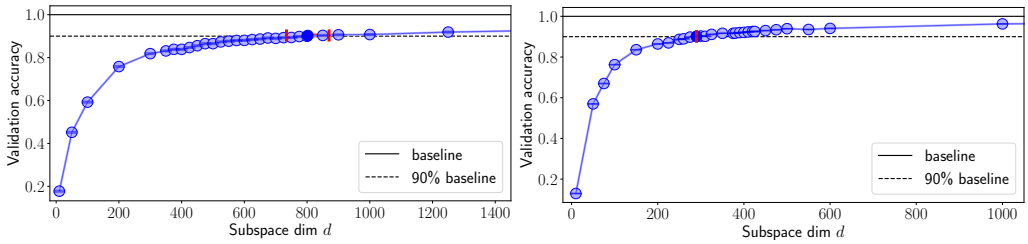


Figure 2: Performance (validation accuracy) *vs.* subspace dimension d for two networks trained on MNIST: **(left)** a 784–200–200–10 fully-connected (FC) network ($D = 199{,}210$) and **(right)** a convolutional network, LeNet ($D = 44{,}426$). The solid line shows performance of a well-trained direct (FC or conv) model, and the dashed line shows the 90% threshold we use to define d_{int90}. The standard derivation of validation accuracy and measured d_{int90} are visualized as the blue vertical and red horizontal error bars. We oversample the region around the threshold to estimate the dimension of crossing more exactly. We use one-run measurements for d_{int90} of 750 and 290, respectively.

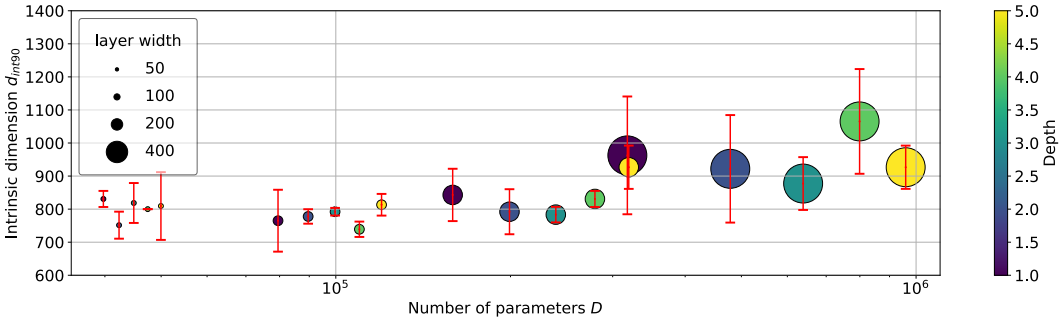


Figure 3: Measured intrinsic dimension d_{int90} vs number of parameters D for 20 FC models of varying width (from 50–400) and depth (number of hidden layers from 1–5) trained on MNIST. The red interval is the standard derivation of masured d_{int90}. Though the number of native parameters varies by a factor of 24.1, d_{int90} varies by only 1.33, with much of that factor possibly due to noise, showing that d_{int90} is a fairly robust measure across a model family and that each extra parameter ends up adding an extra dimension directly to the redundancy of the solution.

Our method differs from previous neural networks compression methods in the following aspects. (i) While it has previously been appreciated that large nets waste parameters (Dauphin & Bengio, 2013) and in general that weights contain redundancy (Denil et al., 2013) that can be exploited for post-hoc compression (Wen et al., 2016), this paper's method constitutes a much simpler approach to compression, where training happens once, end-to-end, and where any parameterized model is an allowable base model. (ii) Unlike layerwise compression models (Denil et al., 2013; Wen et al., 2016), we operate in the entire parameter space, which could work better or worse, depending on the network. (iii) Compared to methods like that of Louizos et al. (2017), who take a Bayesian perspective and consider redundancy on the level of groups of parameters (input weights to a single neuron) by using group-sparsity-inducing hierarchical priors on the weights, our approach is simpler but not likely to lead to compression as high as the levels they attain. (iv) Our approach only reduces the number of degrees of freedom, not the number of bits required to store each degree of freedom, e.g. by quantizing weights (Han et al., 2016). Both approaches could be combined. (v) There is a beautiful array of papers on compressing networks such that they also achieve computational savings during the forward pass (Wen et al., 2016; Han et al., 2016; Yang et al., 2015); subspace training does not speed up execution time during inference. (vi) Finally, note the relationships between weight pruning, weight tying, and subspace training: weight pruning is equivalent to finding, post-hoc, a subspace that is orthogonal to certain axes of the full parameter space and that intersects those axes at the origin. Weight tying, e.g. by random hashing of weights into buckets (Chen et al., 2015), is equivalent to subspace training where the subspace is restricted to lie along the equidistant "diagonals" between any axes that are tied together.

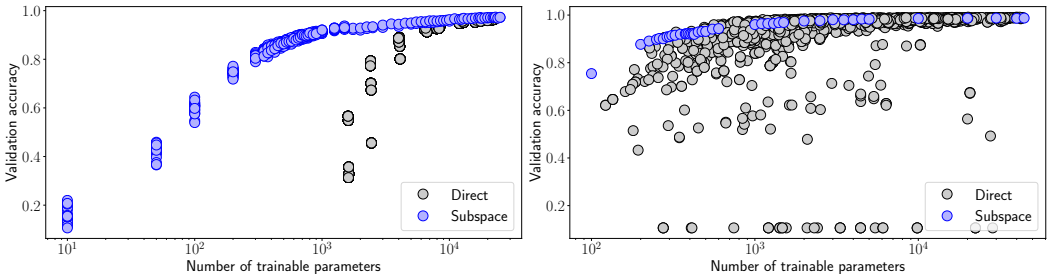


Figure 4: Performance *vs.* number of trainable parameters for **(left)** FC networks and **(right)** convolutional networks trained on MNIST. Randomly generated direct networks are shown (gray circles) alongside all random subspace training results (blue circles) from the sweep shown in Fig. **??**. FC networks show a persistent gap in dimension, suggesting general parameter inefficiency of FC models. The parameter efficiency of convolutional networks varies, as the gray points can be significantly to the right of or close to the blue manifold.

Robustness of intrinsic dimension. Next, we ask how intrinsic dimension varies across FC networks of varying number of layers and varying layer width. Fig. S6 in the Supplementary Material shows performance *vs.* subspace dimension plots in the style of Fig. 2 for a grid search of networks with number of hidden layers L chosen from $\{1, 2, 3, 4, 5\}$ and width W chosen from $\{50, 100, 200, 400\}$. All curves are nearly identical! The intrinsic dimension changes little even as models grown in width or depth; the striking conclusion is that every extra dimension added to D just ends up adding a dimension to the redundancy of the solution. Often the most accurate directly trained models for a problem have far more parameters than needed (Zhang et al., 2017); this may be because they are just easier to train, and our observation suggests a reason why: with larger models, solutions have greater redundancy and in a sense "cover" more of the space. The very similar d_{int90} values for all 20 networks are shown in Fig. 3 plotted against the native dimension D of each network; as one can see, D changes by a factor of 24.1 between the smallest and largest networks, but d_{int90} changes over this range by a factor of only 1.33. Note that for this experiment, we used a global baseline: 100% accuracy, to compare simply and fairly across all models.[3]

Are random subspaces really more parameter-efficient for FC nets? One might wonder to what extent claiming 750 parameters is meaningful given that performance achieved is far lower than a state of the art network trained on MNIST. With such a low bar for performance, could a directly trained network with a comparable number of trainable parameters be found that exhibits the same performance? We generated 1000 smaller networks (depth randomly chosen from $\{1, 2, 3, 4, 5\}$ and layer width from $\{2, 3, 5, 8, 10, 15, 20, 25\}$) in an attempt to find high-performing small FC networks, but as Fig. 4 (left) shows, a $10\times$ gap still exists between the subspace dimension and the smallest direct FC network showing 90% of baseline performance.

Measuring d_{int90} on convolutional networks. We also measure d_{int90} of a convolutional network, LeNet (D=44,426), showing validation accuracy *vs.* subspace dimension d results in Fig. 2 (right). We find $d_{\text{int90}} = 290$, or a compression rate to 0.5%. As with the FC case above, we also do a sweep of random networks, but notice that the performance gap of convnets between direct and subspace training methods becomes closer for fixed budgets, *i.e.,* the number of trainable parameters. Further, the performance of direct training varies significantly, depending on the extrinsic design of convet architectures. We interpret these results in terms of the Minimum Description Length below.

Relationship between Intrinsic Dimension and Minimum Description Length (MDL). As discussed above, the random subspace training method leads naturally to a compressed representation of a network, where only d floating point numbers must be stored. By further assuming that all trainable parameters require the same bits, we can consider this d as a upper bound on the MDL of the problem solution. We cannot yet conclude the extent to which this bound is loose or tight,

[3]See Sec. S5 for similar results obtained using instead 20 separate baselines for each of the 20 models.

Table 1: Measured intrinsic dimension d_{int90} on various problems.

Problems	**MNIST**		**MNIST (Shuf Pixels)**		**CIFAR**		**ImageNet**
Network	**FC**	**LeNet**	**FC**	**LeNet**	**FC**	**LeNet**	**SqueezeNet**
Parameter Dim. D	199,210	44,426	199,210	44,426	656,810	62,006	1,248,424
Intrinsic Dim. d_{int90}	750	290	750	1400	9K	2.9K	$>$ 500K

but it probably varies by problems. However, to the extent that it is tighter than previous bounds (*e.g.,* just the number of parameters D) and to the extent that it is correlated with the actual MDL, as mentioned by Rissanen (1978) and developed by Hinton & Van Camp (1993), we can use this interpretation to judge which solutions more well-suited to the problem in a principled way. Note that although this approach is related to a rich body of work on estimating the "intrinsic dimension of a dataset" (Camastra & Vinciarelli, 2002; Kégl, 2003; Fukunaga & Olsen, 1971; Levina & Bickel, 2005; Tenenbaum et al., 2000), one must be careful to distinguish between estimating the number of principle dimensions of variability in a dataset *vs.* the number of bits necessary to represent a dataset. This is related to representing the data distribution, $p(X)$ *vs.* the number of bits necessary to represent the conditional $p(y|X)$ (as we measure here for supervised models).

Thus, there is some rigor behind our intuitive assumption that we may declare LeNet a *better* model than an FC network for image classification on MNIST, because its intrinsic dimension is lower (d_{int90} of 290 vs 750). In this particular case this leads to a predictable conclusion, but for the countless less well-studied datasets than MNIST, having a simple method of approximating MDL may prove extremely useful for guiding model exploration.

Are convnets always better on MNIST? Measuring d_{int90} on shuffled data. Zhang et al. (2017) provocatively showed that large networks normally thought to generalize well can nearly as easily be trained to memorize entire training sets with randomly assigned labels or with input pixels provided in random order. Consider two networks: one trained on a real, non-shuffled dataset and an identically sized one trained with shuffled pixels or labels. The networks externally are very similar: the training loss may even be identical at the final epoch. However, the intrinsic dimension of each may be measured to expose the differences in problem difficulty. When training on a dataset of **shuffled pixels** , the intrinsic dimension of an FC network remains the same at 750 (as it must). But the intrinsic dimension of a convnet increases from 290 to 1400 — even higher than an FC network. While convnets are better suited to classifying digits given images with natural, local structure, when this structure is removed, violating convolutional assumptions, we can measure that the model's inappropriate assumptions now require even more degrees of freedom to model the underlying distribution. When training on MNIST with **shuffled labels** , we redefine our measure of d_{int90} relative to training accuracy, as validation accuracy necessarily remains at chance. We find that memorizing random labels on the MNIST dataset requires a very high dimension, $d_{\text{int90}} = 190,000$, or 3.8 floats per memorized label. Sec. S5.3 gives a few further results, in particular that the more labels are memorized, the more efficient memorization is (in terms of floats per label). Thus while generalization performance to an unseen validation set is still obviously 0, "generalization" *within* a training set may occur as networks build shared infrastructure for memorizing labels.

3.2 CIFAR-10 AND IMAGENET

We run further experiments on CIFAR-10 (Krizhevsky & Hinton, 2009) and ImageNet (Russakovsky et al., 2015). When attempting to scale beyond MNIST-sized networks — those with D on the order of 200k and d on the order of 1k — we find it necessary to use more efficient methods of generating and projecting from random subspaces. In the case of ImageNet the situation is even severer, as the direct network can easily go beyond millions of parameters. In Sec. S7, we describe and characterize scaling properties of three methods of projection: dense matrix projection, sparse matrix projection (Li et al., 2006), and the beautifully efficient Fastfood transform (Le et al., 2013). We generally use the sparse projection method to train networks on CIFAR, and the Fastfood transform for ImageNet.

Full results on CIFAR10 are given in Sec. S8, and ImageNet in Sec. S9. Measured d_{int90} values are given in Table 1. To summarize, for CIFAR-10 we find qualitatively similar results to MNIST, but with generally higher dimension (8k *vs.* 750 for FC and 2.9k *vs.* 290 for LeNet). Fixing the problem,

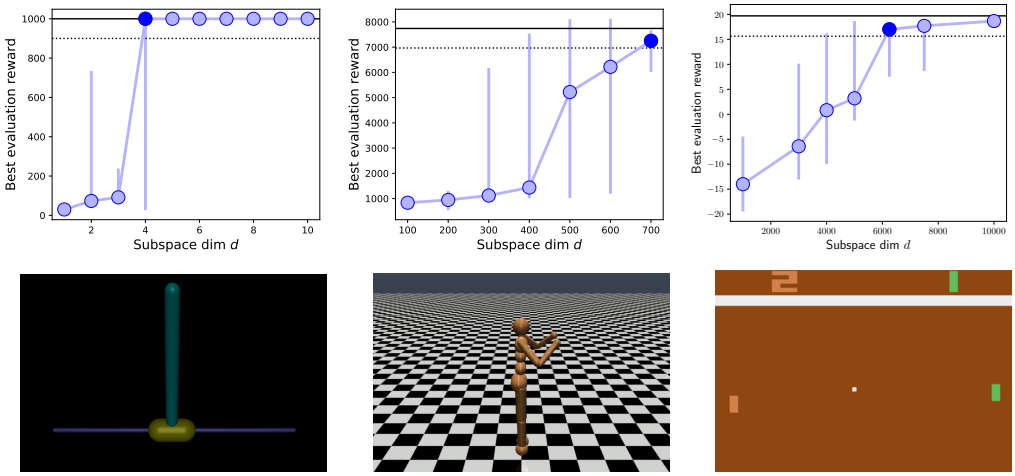


Figure 5: Results using the policy-based ES algorithm to train agents on **(left column)** `InvertedPendulum−v1`, **(middle column)** `Humanoid−v1`, and **(right column)** `Pong−v0`. We find that the inverted pendulum task is surprisingly easy, with $d_{\text{int100}} = d_{\text{int90}} = 4$, meaning that only four parameters are needed to perfectly solve the problem (see Stanley & Miikkulainen (2002) for a similarly small solution found via evolution). The walking humanoid task is more difficult: the solutions are found reliably by dimension 700, the similar complexity required in modeling MNIST with a FC network, and far less than modeling CIFAR-10 with a convnet. Finally, to play Pong on Atari (directly from pixels) requires a network trained in a 6k subspace, making it on the same order of modeling CIFAR-10.

we note d_{int90} can be used as a quantative metric to study the fitness of different architectures for a specific problem. For example, to achieve $>50\%$ validation accuracy on CIFAR10, FC, LeNet and ResNet approximately requires $d_{\text{int90}} = 8$k, 2.9k and 1k, respectively. Due to deadline pressures and memory issues, training on ImageNet has not yet given a reliable estimate for d_{int90} except that it is over 500k.

3.3 Reinforcement Learning environments

Measuring intrinsic dimension allows us to perform the comparison across the divide between supervised learning and reinforcement learning. Below we measure the intrinsic dimensionality of three control tasks of varying difficulties using both value-based and policy-based algorithms. The value-based algorithm we evaluate is the Deep Q-Network (DQN) (Mnih et al., 2013), and the policy-based algorithm is Evolutionary Strategies (ES) (Salimans et al., 2017). Training details are given in Sec. S6.2. For all tasks, performance is defined as the maximum-attained (over training iterations) mean evaluation reward (averaged over 30 evaluations for a given parameter setting). The solid points in the plots correspond to the (noisy) median observed performance values for runs with a given d, and the associated vertical uncertainty intervals show the maximum and minimum observed performance values. The dotted horizontal line corresponds to the usual 90% baseline derived from the best directly-trained network (the solid horizontal line).

In Fig. 5, we provide the main results of ES for three tasks: `InvertedPendulum−v1`, `Humanoid−v1` in MuJoCo (Todorov et al., 2012), and `Pong−v0` in Atari. Please see Sec. S6.2 for more complete ES results, and Sec. S6.1 for DQN results.

4 Conclusions

In this paper, we have defined the intrinsic dimension of objective landscapes to approximately quantatify the difficulty of problems, or the fitness of architectures. When intrinsic dimensions are much lower than the direct parameter dimension, it enables effective network compression; When intrinsic dimension are similar to that of the best tuned models, it endorses those models as well-suited to the problem. A simple method — random subspace training — is proposed to approximate the intrinsic dimension. We discuss future directions in Section S11.

ACKNOWLEDGMENTS

[Removed for blind review.]

REFERENCES

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Francesco Camastra and Alessandro Vinciarelli. Estimating the intrinsic dimension of data with a fractal-based method. *IEEE Transactions on pattern analysis and machine intelligence*, 24(10): 1404–1407, 2002.

Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. *CoRR*, abs/1504.04788, 2015. URL http://arxiv.org/abs/1504.04788.

Yann Dauphin and Yoshua Bengio. Big neural networks waste capacity. *CoRR*, abs/1301.3583, 2013. URL http://arxiv.org/abs/1301.3583.

Yann Dauphin, Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2572, 2014. URL http://arxiv.org/abs/1406.2572.

Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pp. 2148–2156, 2013.

Keinosuke Fukunaga and David R Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, 100(2):176–183, 1971.

Raja Giryes, Guillermo Sapiro, and Alexander M Bronstein. Deep neural networks with random gaussian weights: a universal classification strategy? *IEEE Trans. Signal Processing*, 2016.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Artificial Intelligence and Statistics*, 2010.

Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. Qualitatively characterizing neural network optimization problems. In *International Conference on Learning Representations (ICLR 2015)*, 2015. URL http://arxiv.org/abs/1412.6544.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *CVPR*, 2015.

Geoffrey E Hinton and Drew Van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13. ACM, 1993.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Lukasz Kaiser, Aidan N. Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *CoRR*, abs/1706.05137, 2017. URL http://arxiv.org/abs/1706.05137.

Balázs Kégl. Intrinsic dimension estimation using packing numbers. In *Advances in neural information processing systems*, pp. 697–704, 2003.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114. URL `http://www.pnas.org/content/114/13/3521.abstract`.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Technical report*, 2009.

Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood-approximating kernel expansions in loglinear time. In *ICML*, 2013.

Elizaveta Levina and Peter J Bickel. Maximum likelihood estimation of intrinsic dimension. In *Advances in neural information processing systems*, pp. 777–784, 2005.

Ping Li, T. Hastie, and K. W. Church. Very sparse random projections. In *KDD*, 2006.

C. Louizos, K. Ullrich, and M. Welling. Bayesian Compression for Deep Learning. *ArXiv e-prints*, May 2017.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. *ArXiv e-prints*, December 2013.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

T. Salimans, J. Ho, X. Chen, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *ArXiv e-prints*, March 2017.

Kenneth Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1476–1483, 2015.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *ICLR*, 2017.

SUPPLEMENTARY INFORMATION FOR: MEASURING THE INTRINSIC DIMENSION OF OBJECTIVE LANDSCAPES

S5 ADDITIONAL MNIST RESULTS

S5.1 RESULTS FOR ALL 20 FC NETWORKS

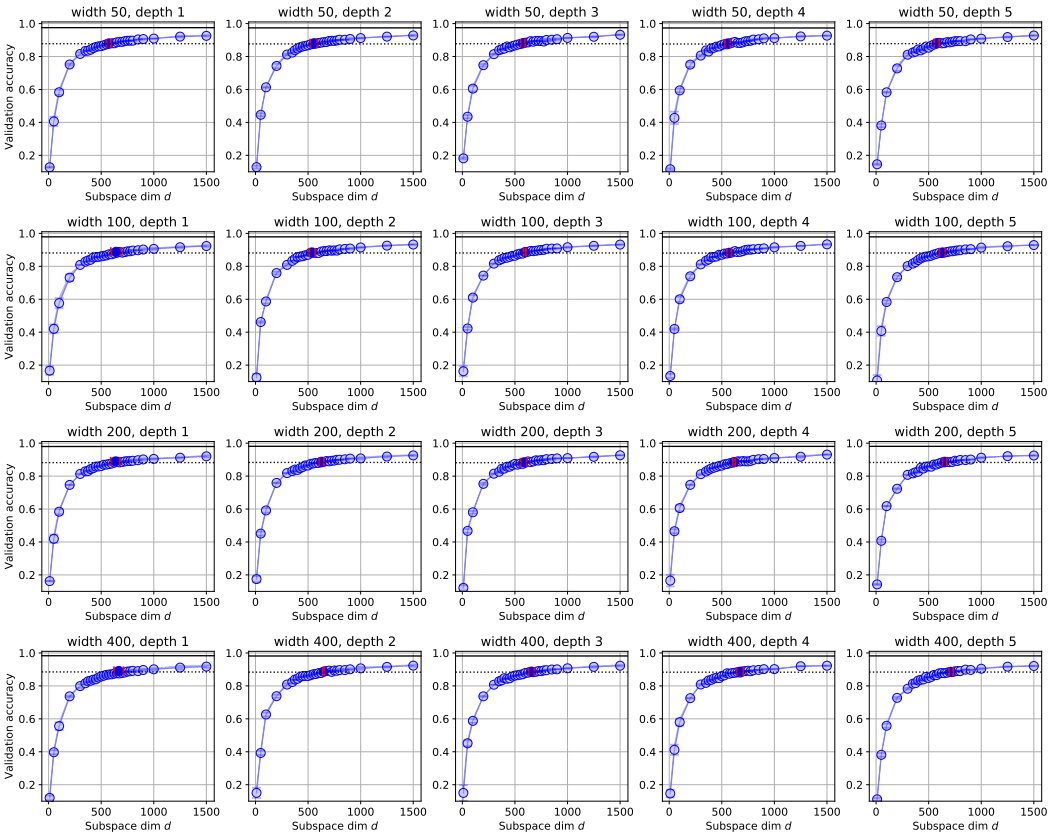


Figure S6: MNIST FC network: validation accuracy vs subspace dimension for a sweep over network depth and width.

We report validation accuracy of different subspace dimensions on all 20 FC networks. The number of hidden layers is chosen from {1,2,3,4,5} and width of each hidden layer is chosen from {50,100,200,400}. The results on 3 runs are considered, and bootstrap at each d up to 50 samples. The means are plot as the dots, and standard derivations of the accuracy are visualized as the vertical blue error bars. The standard derivations of the measured intrinsic dimension d_{int90} is visualized the horizontal red error bars. Note that the variance of both accuarcy and measured d_{int90} for a given hyper-parameter setting are very small, and the mean of performance monotonically increases (very similar to one run result) as d increases. It indicates the luckiness in random projection has a little impact on the quality of solutions, while the subspace dimensionality has a great impact on the quality of solutions. Hence, we can rely on one run result to compute the intrinsic dimension, though slightly more accurate solutions can be obtained via multiple runs.

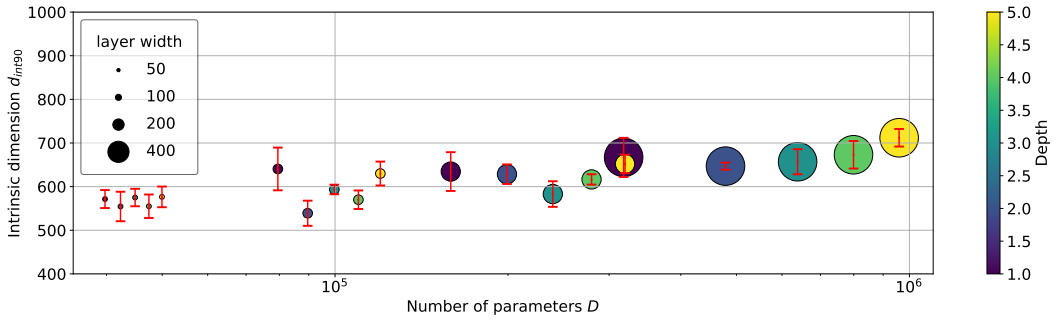
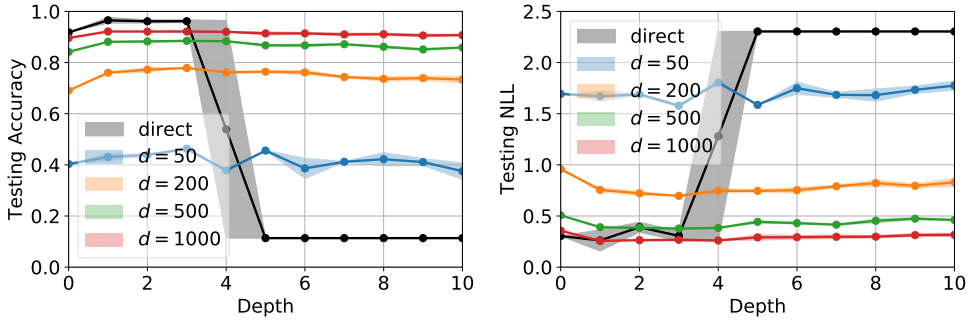


Figure S7: Measured intrinsic dimension d_{int90} vs number of parameters D for 20 models of varying width (from 50 - 400) and depth (number of hidden layers from 1-5) trained on MNIST. The vertical red interval is the standard derivation of measured d_{int90}. As opposed to Fig. 3, which used a global, shared baseline across all models, here a per-model baseline is used. The number of native parameters varies by a factor of 24.1, but d_{int90} varies by only 1.42. The per-model baseline results in higher measured d_{int90} for larger models because they have a higher baseline performance than the shallower models.



(a) Accuracy

(b) Negative Log Likelihood (NLL)

Figure S8: Results of subspace training versus the number of layers in a fully connected network trained on MNIST. The direct method always fail to converge when $L > 5$, while our subspace training yields stable performance across depths.

S5.2 Training Stability

A minor but interesting result is that random subspace training can in some cases make optimization more stable. Fig. S8 shows training results for FC networks with up to 10 layers. SGD with step 0.1, and ReLUs with He initialization is used. Multiple networks failed at depths 4, and all failed at depths higher than 4, despite and activiation function and initialization designed to make learning stable (He et al., 2015). Because each random basis vector projects across all D direct parameters, the optimization problem may be far better conditioned. A related potential downside is that projecting the D parameters which may have widely varying scale could result in ignoring parameter dimensions with tiny gradients. This situation is similar to that faced by methods like SGD, but ameliorated by RMSProp, Adam, and other methods that can rescale individual step sizes to account for individual parameter scales. Though convergence seems robust, further work may be needed to improve network amenability to subspace training: for example by ensuring direct parameters are similarly scaled by clever construction or by inserting a pre-scaling layer between the projected subspace and the direct parameters themselves.

S5.3 Additional Details on Shuffled MNIST Datasets

There are two different shuffled MNIST datasets:

- **The shuffled pixel dataset**: the label for each example remains the same as the normal dataset, but a random permutation of pixels is chosen once and then applied to all images in the training and test sets. FC networks solve the shuffled pixel datasets exactly as easily as the base dataset, because there is no priviliged ordering of input dimension in FC networks – all orderings are equivalent.
- **The shuffled label dataset**: the images remain the same as the base dataset, but training labels are randomly shuffled for the entire training set. Here, as in (Zhang et al., 2017), we only evaluate training accuracy, as test set accuracy remains forever at chance level (the training set X and y convey no information about test set $p(y|X)$, because test set labels are shuffled independent of the training set).

On the full shuffled label MNIST dataset (50k images), we trained an FC network ($L = 5, W = 400$, which had $d_{\text{int90}} = 750$ on stadard MNIST), it yields $d_{\text{int90}} = 190$k. We can interpret this as requiring 3.8 floats to memorize each random label (at 90% accuracy). Curious how this scales with dataset size, we estimated d_{int90} on shuffled label versions of MNIST at different scales and found fairly interesting results, shown in Table S2 and Fig. S9. Futhtermore, 10%, 50%, 100% of the shuffled label MNIST dataset was tested, and d_{int90} increased to 90k, 130k, and 190k, respectively (see Fig. S9). This is consistent with our intuition that the problem becomes harder, as more "random pairs" are required to memorize via the neural networks. As the dataset memorized becomes smaller, the number of floats required to memorize each label becomes larger. The best interpretation is not yet clear, but one possible interpretation is that networks required to memorize large training sets make use of shared machinery for memorization. In other words, though generalization performance to a validation set is still obviously 0, generalization *within* a training set is non-negligible.

Fraction of MNIST training set	$d_{\text{int}}90$	floats per label
100%	190k	3.8
50%	130k	5.2
10%	90k	18.0

Table S2: d_{int90} measured on a dataset memorization task with different dataset sizes

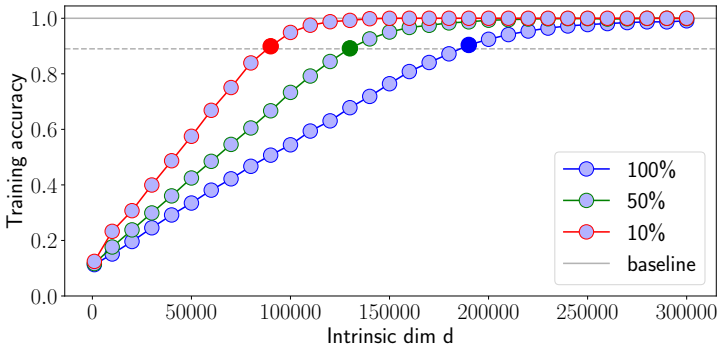


Figure S9: Training accuracy of FC (W=400, L=5) with different subspace dimension d on 100%, 50%, 10% data of shuffled label version of MNIST.

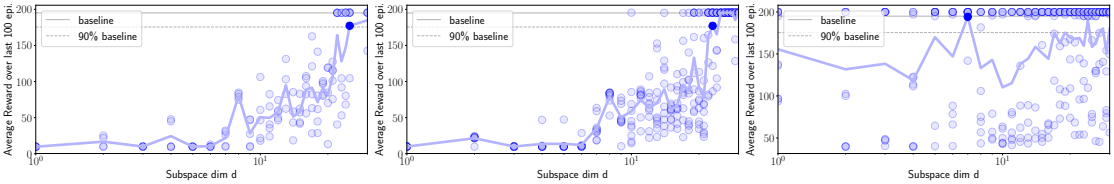
Interestingly, for MNIST with shuffled labels, it is very difficult to reach high training accuracy using direct training method. However, subspace training is stable to excellent performance as d increases.

S5.4 THE ROLE OF OPTIMIZERS

We investigate the impact of stochastic optimization methods on the intrinisc dimension, using FCs on standard MNIST dataset. Two classic optimizers are used: SGD (0.1) and Adam (0.001). The intrinsic dimension d_{int90} are reported in Figure S15 (a) (b).

S6 ADDITIONAL REINFORCEMENT LEARNING RESULTS AND DETAILS

S6.1 DQN EXPERIMENTS



(a) CartPole ($d_{\text{int90}} = 25$) (b) Pole ($d_{\text{int90}} = 23$) (c) Cart ($d_{\text{int90}} = 7$)

Figure S10: Average reward of 3 CartPole environments using DQN.

DQN on Cartpole We start with a simple classic control game `CartPole−v0` in OpenAI Gym (Brockman et al., 2016). The full game ends when one of two failure conditions is satisfied: the cart moves more than 2.4 units from the center, or the pole is more than 15 degrees from vertical. The system is controlled by applying a force of LEFT or RIGHT to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep. We created two potentially easier environments called `Pole` and `Cart`, by removing one of failure conditions respectively. DQN is used, where the value netwowrk is parameterized by an FC ($L = 2, W = 400$). Five runs are tested for each subspace d, the mean is used to computed d_{int90}, where the baseline is set as 195.0[4]. The results are shown in Figure S10. The intrinsic dimension for CartPole, Pole and Cart is $d_{\text{int90}} = 25$, 23 and 7, respectively. It quantifies the difficulty of each part of the full game: driving a cart in a given interval is much easier than keeping a pole straight.

S6.2 ES COMPLETE RESULTS

ES Training Details The hyperparmeters for training the 3 RL tasks using ES are in Table S3.

	ℓ_2 penalty	Adam LR	ES σ	Iterations
`InvertedPendulum−v1`	1×10^{-8}	3×10^{-1}	2×10^{-2}	1000
`Humanoid−v1`	5×10^{-3}	3×10^{-2}	2×10^{-2}	2000
`Pong−v0`	5×10^{-3}	3×10^{-2}	2×10^{-2}	500

Table S3: Hyperparameters used in training RL tasks using ES. σ refers to the parameter perturbation noise used in ES. Default Adam parameters of $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-7}$ were used.

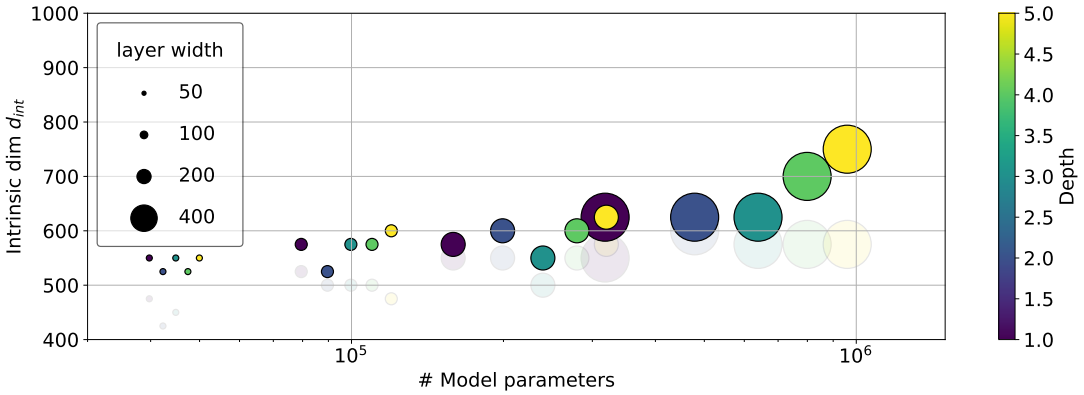
S6.2.1 INVERTED PENDULUM

The `InvertedPendulum−v1` environment uses the MuJoCo physics simulator (Todorov et al., 2012) to instantiate the same problem as `CartPole−v0` in a realistic setting. We expect that even with more accurate environment dynamics, as well as a different RL algorithm – evolutionary strategies (ES) – the intrinsic dimensionality should be similar. As seen in Fig. 5, the measured intrinsic dimensionality $d_{\text{int90}} = 4$ is of the same order of magnitude, but smaller. Interestingly, although the environment dynamics are more complex than in `CartPole−v0`, using ES rather than DQN seems to induce a simpler objective landscape.

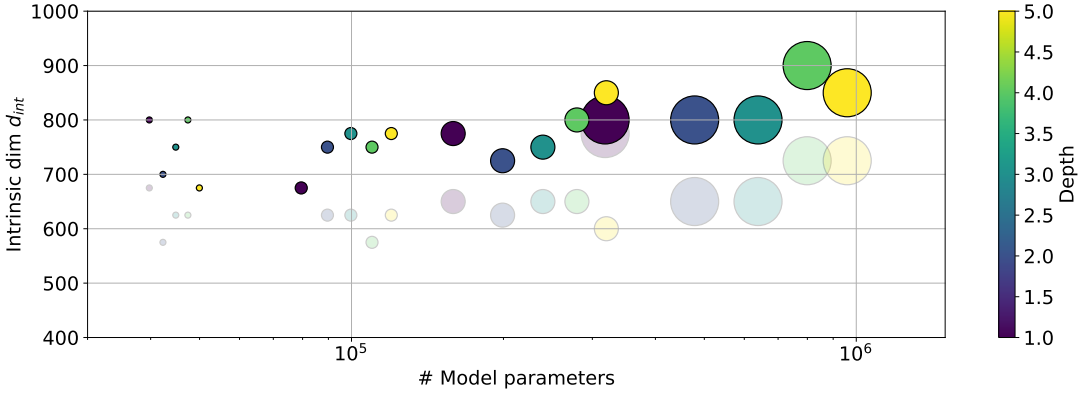
S6.2.2 LEARNING TO WALK

A more challenging problem is `Humanoid−v1` in MuJoCo simulator. Intuitively, one might believe that learning to walk is a more complex task than classifying images. Our results show the contrary – that the learned intrinsic dimensionality of $d_{\text{int90}} = 700$ is similar to that of MNIST on a fully-connected network ($d_{\text{int90}} = 650$) but significantly less than that of even a convnet trained on CIFAR10 ($d_{\text{int90}} = 2,500$). Fig. 5 shows the full results. Interestingly, we begin to see training runs reach the threshold as early as $d = 400$, with the median performance steadily increasing with d.

[4] `CartPole−v0` is considered as "solved" when the average reward over the last 100 episodes is 195



(a) Global baseline



(b) Individual baseline

Figure S11: The role of optimizers. The transparent dots indicate SGD results, and solid dots indicate Adam results. Adam generally yields higher intrinsic dimensions because higher baselines are achieved, especially when individual baselines in (b) are used.

S6.2.3 Atari Pong

Finally, using a base convnet of approximately $D = 1M$ in the `Pong-v0` pixels-to-actions environment (using 4-frame stacking). The agent receives an image frame (size of $210 \times 160 \times 3$) and the action is to move the paddle UP or DOWN. We were able to determine $d_{\text{int90}} = 6,000$.

S7 Three Methods of Random Projection

Scaling the random subspace training procedure to large problems requires an efficient way to map from $\mathbb{R}^d$ into a random d-dimensional subspace of $\mathbb{R}^D$ that does not necessarily include the origin. Algebraically, we need to left-multiply a vector of parameters $v \in \mathbb{R}^d$ by a random matrix $M \in \mathbb{R}^{D \times d}$, whose columns are orthonormal, then add an offset vector $\theta_0 \in \mathbb{R}^D$. If the low-dimensional parameter vector in $\mathbb{R}^d$ is initialized to zero, then specifying an offset vector is equivalent to choosing an initialization point in the original model parameter space $\mathbb{R}^D$.

A naïve approach to generating the random matrix M is to use a dense $D \times d$ matrix of independent standard normal entries, then scale each column to be of length 1. The columns will be approximately orthogonal if D is large because of the independence of the entries. Although this approach is sufficient for low-rank training of models with few parameters, we quickly run into scaling limits because both matrix-vector multiply time and storage of the matrix scale according to $\mathcal{O}(Dd)$. We were able to successfully determine the intrinsic dimensionality of MNIST (d=225) using a LeNet (D=44,426), but were unable to increase d beyond 1,000 when applying a LeNet (D=62,006) to

CIFAR10, which did not meet the performance criterion to be considered the problems intrinsic dimensionality.

Random matrices need not be dense for their columns to be approximately orthonormal. In fact, a method exists for "very sparse" random projections (Li et al., 2006), which achieves a density of $\frac{1}{\sqrt{D}}$. To construct the $D \times d$ matrix, each entry is chosen to be nonzero with probability $\frac{1}{\sqrt{D}}$. If chosen, then with equal probability, the entry is either positive or negative with the same magnitude in either case. The density of $\frac{1}{\sqrt{D}}$ implies $\sqrt{D}d$ nonzero entries, or $\mathcal{O}(\sqrt{D}d)$ time and space complexity. Implementing this procedure allowed us to find the intrinsic dimension of d=2,500 for CIFAR10 using a LeNet mentioned above. Unfortunately, when using Tensorflow's `SparseTensor` implementation we did not achieve the theoretical $\sqrt{D}$-factor improvement in time complexity (closer to a constant 10x). Nonzero elements also have a significant memory footprint of 24 bytes, so we could not scale to larger problems with millions of model parameters and large intrinsic dimensionalities.

We need not explicitly form and store the transformation matrix. The Fastfood transform (Le et al., 2013) was initially developed as an efficient way to compute a nonlinear, high-dimensional feature map $\phi(x)$ for a vector x. A portion of the procedure involves implicitly generating a $D \times d$ matrix with approximately uncorrelated standard normal entries, using only $\mathcal{O}(D)$ space, which can be multiplied by v in $\mathcal{O}(D \log d)$ time using a specialized method. The method relies on the fact that Hadamard matrices multiplied by Gaussian vectors behave like dense Gaussian matrices. In detail, to implicitly multiply v by a random *square* Gaussian matrix M with side-lengths equal to a power of two, the matrix is factorized into multiple simple matrices: $M = HG\Pi HB$, where B is a random diagonal matrix with entries +-1 with equal probability, H is a Hadamard matrix, Π is a random permutation matrix, and G is a random diagonal matrix with independent standard normal entries. Multiplication by a Hadamard matrix can be done via the Fast Walsh-Hadamard Transform in $\mathcal{O}(d \log d)$ time and takes no additional space. The other matrices have linear time and space complexities. When $D > d$, multiple independent samples of M can be stacked to increase the output dimensionality. When d is not a power of two, we can zero-pad v appropriately. Stacking $\frac{D}{d}$ samples of M results in an overall time complexity of $\mathcal{O}(\frac{D}{d} d \log d)$ = $\mathcal{O}(D \log d)$, and a space complexity of $\mathcal{O}(\frac{D}{d} d)$ = $\mathcal{O}(D)$. In practice, the reduction in space footprint allowed us to scale to much larger problems, including the Pong RL task using a 1M parameter convolutional network for the policy function.

Table S4 summarizes the performance of each of the three methods theoretically and empirically.

	Time complexity	Space complexity	D = 100k	D = 1M	D = 60M
Dense	$\mathcal{O}(Dd)$	$\mathcal{O}(Dd)$	0.0169 s	1.0742 s*	4399.1 s*
Sparse	$\mathcal{O}(\sqrt{D}d)$	$\mathcal{O}(\sqrt{D}d)$	0.0002 s	0.0019 s	0.5307 s*
Fastfood	$\mathcal{O}(D \log d)$	$\mathcal{O}(D)$	0.0181 s	0.0195 s	0.7949 s

Table S4: Comparison of theoretical complexity and average duration of a forward+backward pass through M (in seconds). d was fixed to 1% of D in each measurement. D = 100k is approximately the size of an MNIST fully-connected network, and D = 60M is approximately the size of AlexNet. The Fastfood timings are based on a Tensorflow implementation of the Fast Walsh-Hadamard Transform, and could be drastically reduced with an efficient CUDA implementation. Asterisks mean that we encountered an out-of-memory error, and the values are extrapolated from the largest successful run (a few powers of two smaller). For example, we expect sparse to outperform Fastfood if it didn't run into memory issues.

Figure S4 compares the computational time for direct and subspace training (various projections) methods for each update. Our subspace training is more computational expensive, because the subspace training method has to propagate the signals through two modules: the layers of neural networks, and the projection between two spaces. The direct training only propagates signals in the layers of neural networks. We have made efforts to reduce the extra computational cost. For example, the sparse projection only doubles the time cost for a large range of subspace dimensions.

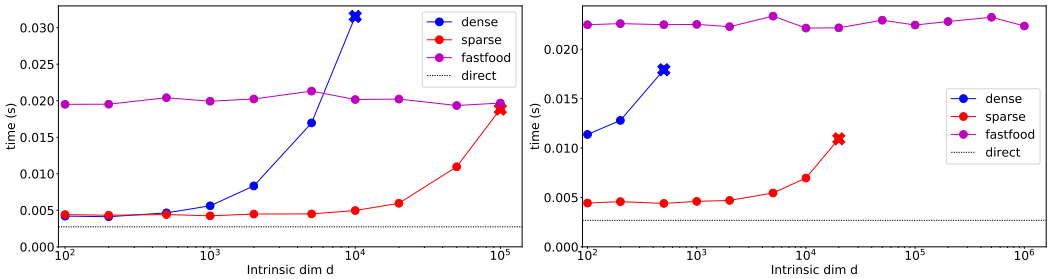
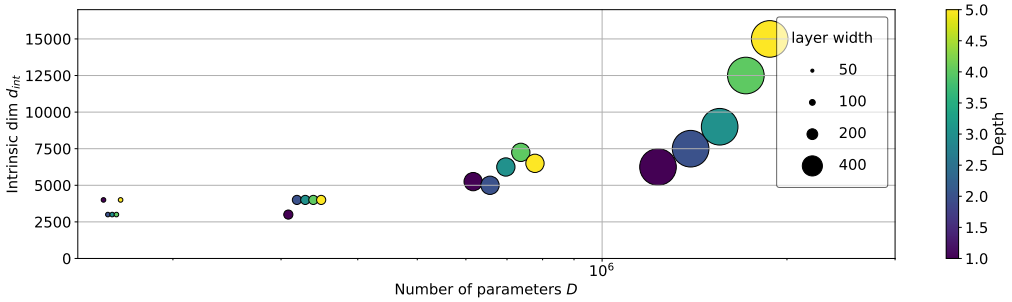
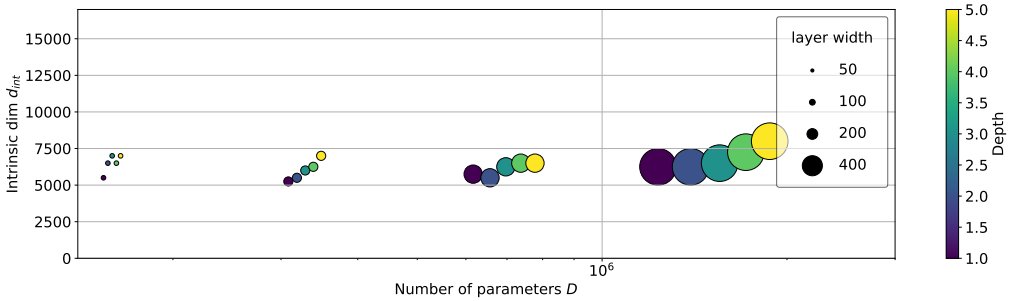


Figure S12: MNIST compute time for direct *vs.* various projection methods for 100K parameters (left) and 1M parameters (right).



(b) Individual baseline



(a) Global baseline

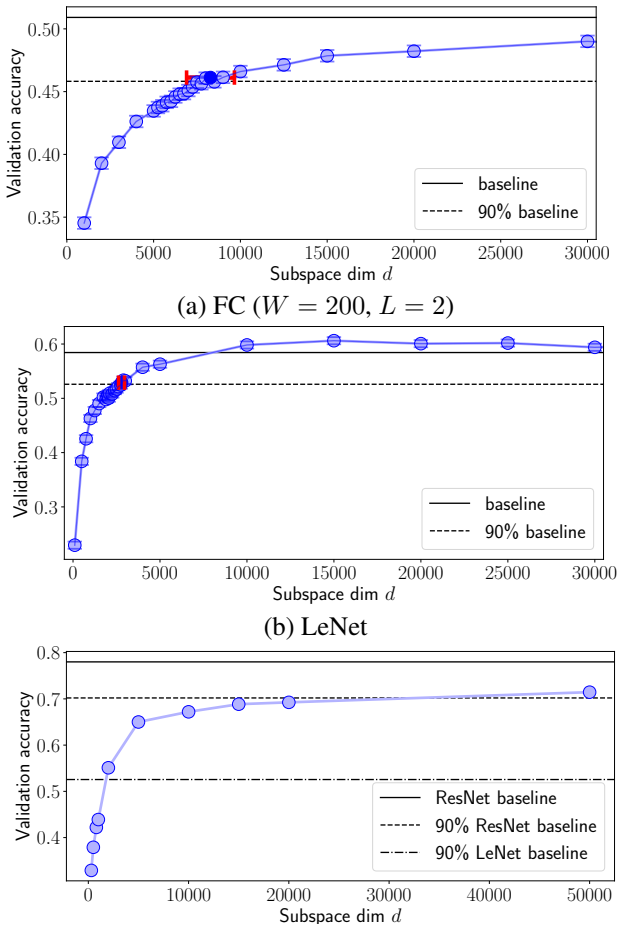
Figure S13: Intrinsic dimension of FC with various width and depth on CIFAR 10 dataset.

S8 ADDITIONAL CIFAR-10 RESULTS

FC networks We consider the CIFAR10 dataset, and test the same set of FC and LeNet architecure as on MNIST. For FC networks, d_{int90} values for all 20 networks are shown in S13 (a) plotted against the native dimension D of each network; D changes by a factor of 12.16 between the smallest and largest networks, but d_{int90} changes over this range by a factor of 5.0. In S13 (b), we also compute the intrinsic dimension wrt to a global baseline: 50% validation accuracy, d_{int90} changes over this range by a factor of 1.52. This indicates that various FCs share similar intrinsic dimension ($d_{\text{int90}} = 5000 \sim 8000$) to achieve the same level of task performance. For LeNet (D=62,006), the validation accuracy *vs.* subspace dimension d is shown in Fig. S14 (a), the corresponding $d_{\text{int90}} = 2900$. It yields a compression rate of 5%, which is 10 times larger than LeNet on MNIST. It shows that CIFAR images are significantly more difficult to be correctly classified than MNIST. In another word, CIFAR is a harder problem than MNIST, especially given the fact that the notion of "problem solved" (baseline performance) is defined as 99% accuracy on MNIST and 58% accuracy on CIFAR. On the CIFAR dataset, as d increases, subspace training tends to overfitting, we will study the role of subspace training as a regularizer below.

ResNet *vs.* LeNet In addition, ResNet exhibits efficient use of parameters with its introduction of residual connections. We are interested to know whether the intrinsic dimension of ResNet on CIFAR is any different. We adopt the smallest 20-layer structure of ResNet with 280K parameters,

and find out in Fig. S14 (b) that it reaches LeNet baseline with $d_{\text{int90}} = 1000 \sim 2000$ (lower than the d_{int90} of LeNet), while takes a larger d_{int90} $(20,000 \sim 50,000)$ to reach reach its own, much higher baseline.



(a) FC ($W = 200$, $L = 2$)

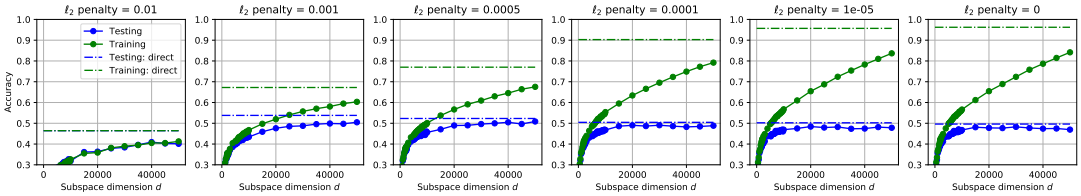
(b) LeNet

(c) ResNet

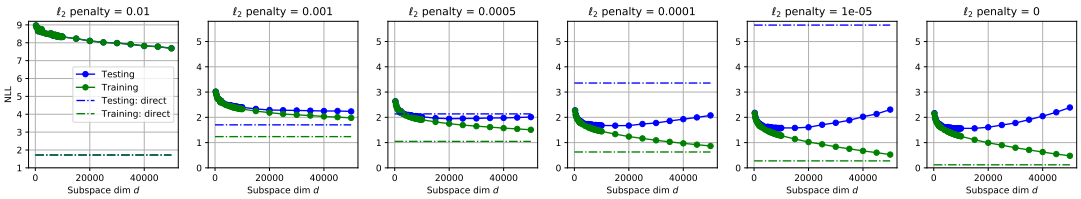
Figure S14: Validation accuracy of an FC network, LeNet and ResNet on CIFAR with different subspace dimension d. In (a)(b), the variance of validation accuracy and measured d_{int90} are visualized as the blue vertical and red horizontal error bars, respectively. Subspace method surpasses the 90% baseline on LeNet at d between 1000 and 2000, 90% of ResNet baseline between 20k and 50k.

The role of regularizers Our subspace training can be considered as a regularization scheme, as it restricts the solution sets. We study and compare its effects with two traditional regularizers with an FC netwrok (L= 2 W=200) on CIFAR10 dataset, including ℓ_2 penalty on the weights (*i.e.,* weight decay) and Dropout.

- ℓ_2 **penalty** Various amount of ℓ_2 penalty $\{10^{-2}, 10^{-3}, 10^{-4}, 5\times10^{-4}, 10^{-5}, 0\}$ are considered. The accuracy and negative log-likelihood (NLL) are reported in Figure S15 (a) (b), respectively. As expected, larger amount of weight decay reduces the gap between training and testing performance for both direct and subspace training methods, and eventually closes the gap (*i.e.,* ℓ_2 penalty = 0.01). Subspace training itself exhibits strong regularization ability, especially when d is small, at which the performance gap beween training and testing is smaller.
- **Dropout** Various dropout rate $\{0.5, 0.4, 0.3, 0.2, 0.1, 0\}$ are considered. The accuracy and NLL are reported in Figure S16. Larger amount of dropout rates reduces the gap between training and testing performance for both direct and subspace training methods. When obserserving testing NLL, subspace training tend to less overfitt the training dataset.
- **Subspace training as implicit regularization** Subspace training method performs implicit regularization, as it restricts the solution set. We visualized the testing NLL in Figure S17. Subspace training method outperforms direct method when d is properly chosen (when ℓ_2 penalty$< 5\times10^{-4}$, or dropout rate < 0.1), suggesting the potential of this method as a better alternative to traditional regularizers. When d is large, the method also overfits the training dataset. Note that the these methods perform regularization in different ways: weight decay enforces the learned weights concentrating around zeros, while subspace training directly reduces the number of dimensions of the solution space.



(a) Accuracy



(b) NLL

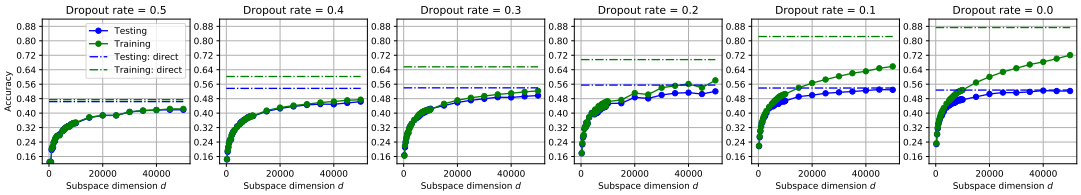
Figure S15: Comparing regularization induced by ℓ_2 penalty and subspace training. Weight decay interacts with d_{int90} since it changes the objective landspaces through various loss functions.

S9 IMAGENET

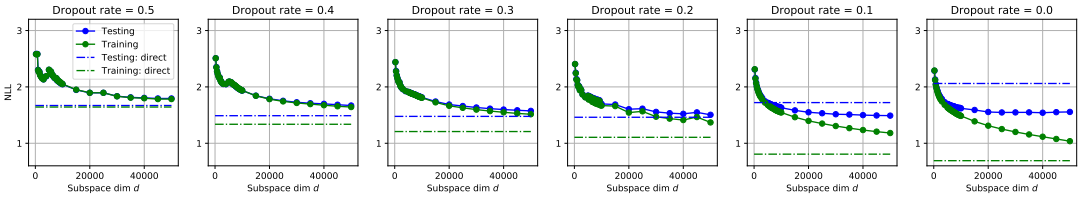
To investigate the applicability of our proposed method to large-scale problems, we take on ImageNet classification. A relatively smaller network, SqueezeNet by Iandola et al. (2016), of parameter size 1.24M, is used. A direct training produces Top-1 accuracy of 55.5%. We vary intrinsic dimension from 300 to 500K, and record the validation accuracies as shown in Fig. S18. The training of each intrinsic dimension takes about 6 to 7 days, while distributed among 4 GPUs.

S10 INVESTIGATION OF CONVOLUTIONAL NETWORKS

Since the learned d_{int90} can be used as a robust measure to study the fitness of neural network architectures for specific tasks, we further apply it to understand the contribution of each component in convolutional networks for image classification task. The convolutional network is a special case of FC network in two aspects: local receptive fields and weight-tying. local receptive fields allows each filter only "looks" at a small, localized region of the image. Weight-tying enforces that each filter shares the same weights.

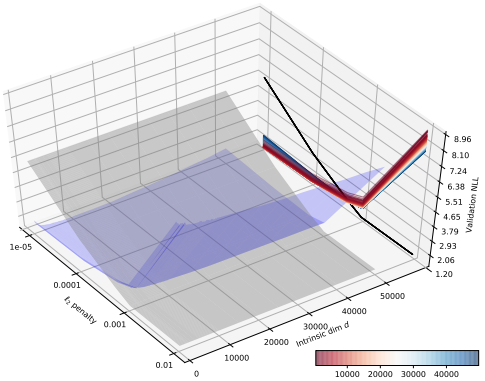


(a) Accuracy

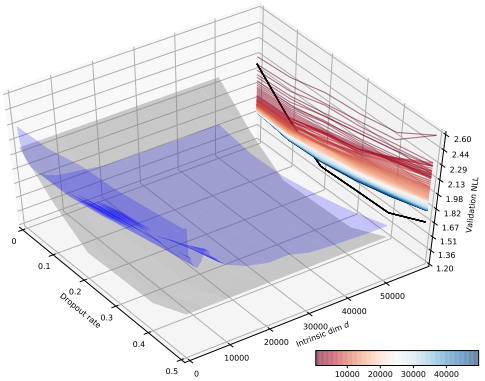


(b) NLL

Figure S16: Comparing regularization induced by Dropout and subspace training. Dropout interacts with d_{int90} since it changes the objective landspaces through randomly removing hidden units of the extrinsic neural networks.



(a) ℓ_2 penalty



(b) Dropout

Figure S17: Comparing regularization induced by ℓ_2 penalty, Dropout and subspace training. The balck surface and line indicate direct training.

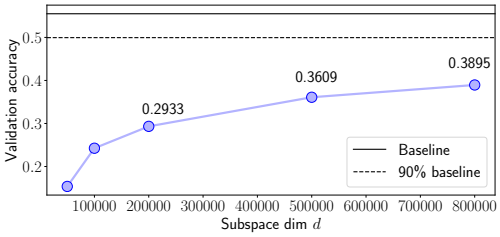


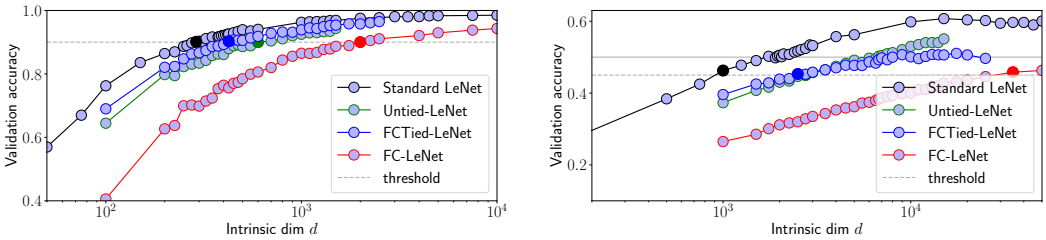
Figure S18: Validation accuracy of SqueezeNet on ImageNet with different d. At $d = 500k$ the accuracy reaches 34.34%, which is not yet past the threshold required to estimate d_{int90}.

This reduces the number of learnable parameters, We did control experiments to investi-

gate the degree of each component contributes. Four variants of LeNet are considered:

- **Standard LeNet** 6 kernels (5×5) – max-pooling (2×2) – 16 kernels (5×5) – max-pooling (2×2) – 120 FC – 84 FC – 10 FC
- **Untied-LeNet** The same architecture with the standard LeNet is employed, except that weights are unshared, *i.e.,* a different set of filters is applied at each different patch of the input. For example in Keras, the `LocallyConnected2D` layer is used to replace the `Conv2D` layer.
- **FCTied-LeNet** The same set of filters is applied at each different patch of the input. we break local connections by applying filters to global patches of the input. Assume the image size is $H \times H$, the architecture is 6 kernels $((2H-1)\times(2H-1))$ – max-pooling (2×2) – 16 kernels $((H-1)\times(H-1))$ – max-pooling (2×2) – 120 FC – 84 FC – 10 FC. The padding type is `same`.
- **FC-LeNet** Neither local connections or tied weights is employed, we mimic LeNet with its FC implementation. The same number of hidden units with the standard LeNet is maintained at each layer.

The results are shown in Fig. S19 (a)(b). We set a crossing-line accuracy (*i.e.,* threshold) for each task, and investigate d_{int90} needed to achieve it. For MNIST and CIFAR, the threshold is 90% and 45%, respectively. For the above LeNet variants, $d_{\text{int90}} = 290, 600, 425, 2000$ on MNIST, and $d_{\text{int90}} = 1000, 2750, 2500, 35000$ on CIFAR. It shows local connections is key to convolutional networks, and the tied-weights mechanism is also important to the excellent performance.



(a) MNIST (b) CIFAR

Figure S19: Validation accuracy of LeNet variants with different subspace dimension d.

S11 SUMMARIZATION OF d_{int90}

We summarize d_{int90} of the objective landspace on different problems and neural network architectures in Table S5 and Fig. S20. "SP" indicates shuffled pixel, and "PL" for shuffled label, and "FC-5" for a 5-layer FC. By fixing the architecture as FC, we see that different intrinsic dimensions are obtained on various problems in different domains. Though the scaling factor of D for FC is small (as a result that we fixed FC), the obtained d_{int90} can change signicantly on different problems. d_{int90} indicates the mimimum number of trainable parameters required to solve the problem, and thus reflects the difficulty level of problems. Similar conclusions can be reached when fixing the architecture as LeNet.

In cases where intrinsic dimension is far smaller than the direct dimension, future work could investigate using much more powerful solvers — *e.g.,* more expensive second order methods — due to the parsimonious parameter length. Recent promising attempts to solve catastrophic forgetting like Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017) have relied on annotating individual parameters in a network with their relative importance; training in random subspaces yields directly a subspace that matters and the dimension — though not orientation — of the subspace that does not, which could enable networks that forget less. Further work could also identify better ways of creating subspaces for reparameterization: here we chose random linear subspaces, but one might carefully construct linear or non-linear subspaces to be even more efficient, or pre-scale or otherwise transform the direct parameters before projecting to or from the subspace to make projections more likely to intersect solutions. Finally, as the field departs from single stack-of-layers image classification models toward larger and more heterogeneous networks (Ren et al., 2015; Kaiser et al.,

2017) often composed of many modules and trained by many losses, methods — like measuring intrinsic dimension — that allow some automatic assessment of model components might provide much-needed greater understanding of individual black-box module properties.

Table S5: Intrinsic dimension of the objective landspaces.

Dataset	Network	D	d_{int90}
MNIST	FC	199210	750
MNIST	LeNet	44426	275
Cifar	FC	1393610	8000
Cifar	LeNet	62006	2900
MNIST-SP	FC	199210	750
MNIST-SP	LeNet	44426	650
MNIST-SL-100%	FC-5	959610	190000
MNIST-SL-50%	FC-5	959610	130000
MNIST-SL-10%	FC-5	959610	90000
ImageNet	SqueezeNet	124842	>500000
CartPole	FC	199210	25
Pole	FC	199210	23
Cart	FC	199210	7
Inverted Pendulum	FC	562	4
Humanoid	FC	166673	700
Atari Pong	ConvNet	1005974	6000

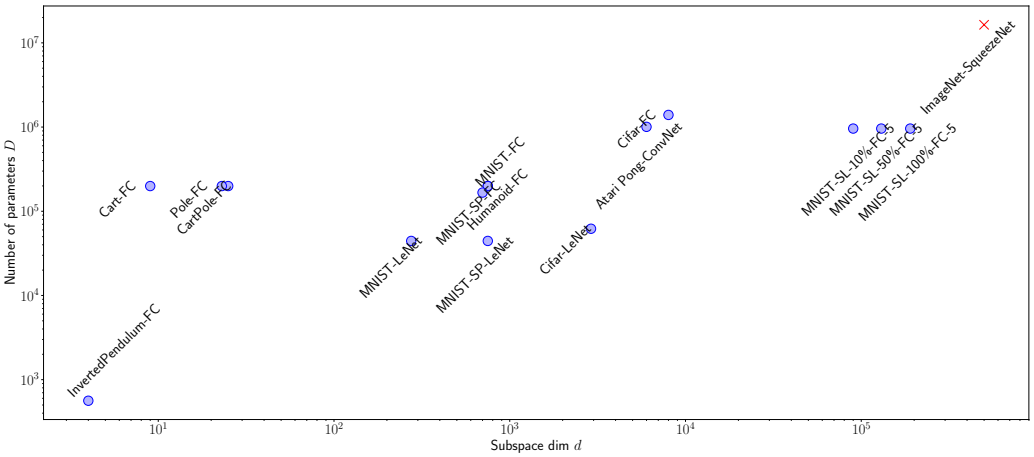


Figure S20: Intrinsic dimension of the objective landspaces.