Diversity Networks

Zelda Mariet and Suvrit Sra
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
zelda@csail.mit.edu,suvrit@mit.edu

Abstract

We introduce Divnet, a flexible technique for learning networks with diverse neurons. Divnet models neuronal diversity by placing a Determinantal Point Process (DPP) over neurons in a given layer. It uses this DPP to select a subset of diverse neurons and subsequently fuses the redundant neurons into the selected ones. Compared with previous approaches, Divnet offers a more principled, flexible technique for capturing neuronal diversity and thus implicitly enforcing regularization. This enables effective auto-tuning of network architecture and leads to smaller network sizes without hurting performance. Moreover, through its focus on diversity and neuron fusing, Divnet remains compatible with other procedures that seek to reduce memory footprints of networks. We present experimental results to corroborate our claims: for pruning neural networks, Divnet is seen to be notably superior to competing approaches.

1 Introduction

Training neural networks requires setting several hyper-parameters to adequate values: number of hidden layers, number of neurons per hidden layer, learning rate, momentum, dropout rate, etc. Although tuning such hyper-parameters via parameter search has been recently investigated by Maclaurin et al. (2015), doing so remains extremely costly, which makes it imperative to develop more efficient techniques.

Of the many hyper-parameters, those that determine the network's architecture are among the hardest to tune, especially because changing them during training is more difficult than adjusting more dynamic parameters such as the learning rate or momentum. Typically, the architecture parameters are set once and for all before training begins. Thus, assigning them correctly is paramount: if the network is too small, it will not learn well; if it is too large, it may take significantly longer to train while running the risk of overfitting. Networks are therefore usually trained with more parameters than necessary, and pruned once the training is complete.

This paper introduces Divnet, a new technique for reducing the size of a network. Divnet decreases the amount of redundancy in a neural network (and hence its size) in two steps: first, it samples a diverse subset of neurons; then, it merges the remaining neurons with the ones previously selected.

Specifically, Divnet models neuronal diversity by placing a Determinantal Point Process (DPP) (Hough et al., 2006) over neurons in a layer, which is then used to select a subset of diverse neurons. Subsequently, Divnet "fuses" information from the dropped neurons into the selected ones through a reweighting procedure. Together, these steps reduce network size (and act as implicit regularization), without requiring any further training or significantly hurting performance. Divnet is fast and runs in time negligible compared to the network's prior training time. Moreover, it is agnostic to other network parameters such as activation functions, number of hidden layers, and learning rates.

For simplicity, we describe and analyze Divnet for feed-forward neural networks, however Divnet is *not* limited to this setting. Indeed, since Divnet operates on a layer fully connected to the following one in a network's hierarchy, it applies equally well to other architectures with fully connected layers. For example, it can be applied without any further modification to Deep Belief Nets and to the fully-connected layers in Convolutional Neural Networks. As these layers are typically responsible for the large majority of the CNNs' memory footprint (Yang et al., 2014), DIVNET is particularly well suited for such networks.

Contributions. The key contributions of this paper are the following:

– Introduction of DPPs as a flexible, powerful tool for modeling layerwise neuronal diversity (§2.1). Specifically, we present a practical method for creating DPPs over neurons, which enables diversity promoting sampling and thereby leads to smaller network sizes.
– A simple but crucial "fusing" step that minimizes the adverse effects of removing neurons. Specifically, we introduce a reweighting procedure for a neuron's connections that transfers the contributions of the pruned neurons to the ones that are retained (§2.2).

The combination of both ideas is called DIVNET. We perform several experiments to validate DIVNET and compare to previous neuron pruning approaches, which DIVNET consistently outperforms. Notably, DIVNET's reweighting strategy benefits other pruning approaches.

Related work. Due to their large number of parameters, deep neural networks typically have a heavy memory footprint. Moreover, in many deep neural network models parameters show a significant amount of redundancy (Denil et al., 2013). Consequently, there has been significant interest in developing techniques for reducing a network's size without penalizing its performance.

A common approach to reducing the number of parameters is to remove connections between layers. In (LeCun et al., 1990; Hassibi et al., 1993), connections are deleted using information drawn from the Hessian of the network's error function. Sainath et al. (2013) reduce the number of parameters by analyzing the weight matrices, and applying low-rank factorization to the final weight layer. Han et al. (2015) remove connections with weights smaller than a given threshold before retraining the network. These methods focus on deleting parameters whose removal influences the network the least, while DIVNET seeks diversity and merges similar neurons; these methods can thus be used in conjunction with ours. Although methods such as (LeCun et al., 1990) that remove connections between layers may also delete neurons from the network by removing all of their outgoing or incoming connections, it is likely that the overall impact on the size of the network will be lesser than approaches such as DIVNET that remove entire neurons: indeed, removing a neuron decreases the number of rows or columns of the weight matrices connecting the neuron's layer to both the previous and following layers.

Convolutional Neural Networks (LeCun et al., 1998) replace fully-connected layers with convolution and subsampling layers, which significantly decreases the number of parameters. However, as CNNs still maintain fully-connected layers, they also benefit from DIVNET.

Closer to our approach of reducing the number of hidden neurons is (He et al., 2014), which evaluates each hidden neuron's importance and deletes neurons with the smaller importance scores. In (Srinivas and Babu, 2015), a neuron is pruned when its weights are similar to those of another neuron in its layer, leading to a weight update procedure that is somewhat similar in idea (albeit simpler) to our reweighting step: where (Srinivas and Babu, 2015) removes neurons with equal or similar weights, we consider the more complicated task of merging neurons that, as a group, perform redundant calculations based on their activations.

Other recent approaches consider network compression without pruning: in (Hinton et al., 2015), a new, smaller network is trained on the outputs of the large network; Chen et al. (2015) use hashing to reduce the size of the weight matrices by forcing all connections within the same hash bucket to have the same weight. Courbariaux et al. (2014) and Gupta et al. (2015) show that networks can be trained and run using limited precision values to store the network parameters, thus reducing the overall memory footprint.

We emphasize that DIVNET's focus on neuronal diversity is orthogonal and complementary to prior network compression techniques. Consequently, DIVNET can be combined, in most cases trivially, with previous approaches to memory footprint reduction.

2 Diversity and redundancy reduction

In this section we introduce our technique for modeling neuronal diversity more formally.

Let $\mathcal{T}$ denote the training data, ℓ a layer of n_ℓ neurons, a_{ij} the activation of the i-th neuron on input t_j, and $v_i = (a_{i1}, \dots, a_{in_\ell})^\top$ the activation vector of the i-th neuron obtained by feeding the training data through the network. To enforce diversity in layer ℓ, we must determine which neurons are computing redundant information and remove them. Doing so requires finding a maximal subset of (linearly) independent activation vectors in a layer and retaining only the corresponding neurons. In practice, however, the number of items in the training set (or the number of batches) can be much larger than the number of neurons in a layer, so the activation vectors $v_1, \dots, v_{n_\ell}$ are likely linearly independent. Merely selecting by the maximal subset may thus lead to a trivial solution that selects all neurons.

Reducing redundancy therefore requires a more careful approach to sampling. We propose to select a subset of neurons whose activation patterns are diverse while contributing to the network's overall computation (i.e., their activations are not saturated at 0). We achieve this diverse selection by formulating the neuron selection task as sampling from a Determinantal Point Process (DPP). We describe the details below.

2.1 Neuronal diversity via Determinantal Point Processes

DPPs are probability measures over subsets of a ground set of items. Originally introduced to model the repulsive behavior of fermions (Macchi, 1975), they have since been used fruitfully in machine-learning (Kulesza and Taskar, 2012). Interestingly, they have also been recently applied to modeling inter-neuron inhibitions in neural spiking behavior in the rat hippocampus (Snoek et al., 2013).

DPPs present an elegant mathematical technique to model diversity: the probability mass associated to each subset is proportional to the determinant (hence the name) of a DPP kernel matrix. The determinant encodes negative associations between variables, and thus DPPs tend to assign higher probability mass to diverse subsets (corresponding to diverse submatrices of the DPP kernel). Formally, a ground set of N items $\mathcal{Y} = \{1, \dots, N\}$ and a probability $\mathcal{P} : 2^{\mathcal{Y}} \to [0, 1]$ such that

$$\mathcal{P}(Y) = \frac{\det(L_Y)}{\det(L + I)}, \tag{1}$$

where L is a N-by-N positive definite matrix, form a DPP. L is called the *DPP kernel*; here, L_Y indicates the $|Y| \times |Y|$ principal submatrix of L indexed by the elements of Y.

The key ingredient that remains to be specified is the DPP kernel, which we now describe.

2.1.1 Constructing the DPP kernel

There are numerous potential choices for the DPP kernel. We found that experimentally a well-tuned Gaussian RBF kernel provided a good balance between simplicity and quality: for instance, it provides much better results that simple linear kernels (obtained via the outer product of the activation vectors) and is easier to use than more complex Gaussian RBF kernels with additional parameters. A more thorough evaluation of kernel choice is future work.

Recall that layer ℓ has activations $v_1, \dots, v_{n_\ell}$. Using these, we first create an $n_\ell \times n_\ell$ kernel L' with bandwidth parameter β by setting

$$L'_{ij} = \exp(-\beta \|v_i - v_j\|^2) \qquad 1 \leq i, j \leq n_\ell. \tag{2}$$

To ensure strict positive definiteness of the kernel matrix L', we add a small diagonal perturbation εI to L' ($\varepsilon = 0.01$). The choice of the bandwidth parameter could be done by cross-validation, but that would greatly increase the training cost. Therefore, we use the fixed choice $\beta = 10/|\mathcal{T}|$, which was experimentally seen to work well.

Finally, in order to limit rounding errors, we introduce a final scaling operation: suppose we wish to obtain a desired size, say k, of sampled subsets (in which case we are said to be using a k-DPP (Kulesza and Taskar, 2011)). To that end, we can scale the kernel $L' + \varepsilon I$ by a factor γ, so that its *expected* sample size becomes k. For a DPP with kernel L, the expected sample size is given by (Kulesza and Taskar, 2012, Eq. 34):

$$\mathbb{E}[|Y|] = \mathrm{Tr}(L(I+L)^{-1}).$$

Therefore, we scale the kernel to $\gamma(L' + \varepsilon I)$ with γ such that

$$\gamma = \frac{k}{n_\ell - k} \cdot \frac{n_\ell - k'}{k'},$$

where k' is the expected sample size for the kernel $L' + \varepsilon I$.

Finally, generating and then sampling from $L = \gamma(L' + \varepsilon I)$ has $\mathcal{O}(n_\ell^3 + n_\ell^2|\mathcal{T}|)$ cost. In our experiments, this sampling cost was negligible compared with the cost of training. For networks with very large hidden layers, one can avoiding the n_ℓ^3 cost by using more scalable sampling techniques (Li et al., 2015; Kang, 2013).

2.2 Fusing redundant neurons

Simply excising the neurons that are not sampled by the DPP drastically alters the neuron inputs to the next layer. Intuitively, since activations of neurons marked redundant are not arbitrary, throwing them away is wasteful. Ideally we should preserve the total information of a given layer, which suggests that we should "fuse" the information from unselected neurons into the selected ones. We achieve this via a reweighting procedure as outlined below.

Without loss of generality, let neurons 1 through k be the ones sampled by the DPP and $v_1, \ldots, v_k$ their corresponding activation vectors. Let w_{ij} be the weights connecting the i-th neuron ($1 \le i \le k$) in the current layer to the j-th neuron in the next layer; let $\widetilde{w}_{ij} = \delta_{ij} + w_{ij}$ denote the updated weights after merging the contributions from the removed neurons.

We seek to minimize the impact of removing $n_\ell - k$ neurons from layer ℓ. To that end, we minimize the difference in inputs to neurons in the subsequent layer before ($\sum_{i \le n_\ell} w_{ij} v_i$) and after ($\sum_{i=1 \le k} \widetilde{w}_{ij} v_i$) DPP pruning. That is, we wish to solve for all neurons in the next layer (indexed by j, $1 \le j \le n_{\ell+1}$):

$$\min_{\widetilde{w}_{ij} \in \mathbb{R}} \left\| \sum\nolimits_{i=1}^{k} \widetilde{w}_{ij} v_i - \sum\nolimits_{i=1}^{n_\ell} w_{ij} v_i \right\|_2 = \min_{\delta_{ij} \in \mathbb{R}} \left\| \sum\nolimits_{i=1}^{k} \delta_{ij} v_i - \sum\nolimits_{i=k+1}^{n_\ell} w_{ij} v_i \right\|_2. \tag{3}$$

Eq. 3 is minimized when $\sum_{i \le k} \delta_{ij} v_i$ is the projection of $\sum_{i>k} w_{ij} v_i$ onto the linear space generated by $\{v_1, \ldots, v_k\}$. Thus, to minimize Eq. 3, we obtain the coefficients α_{ij} that for $j > k$ minimize

$$\left\| v_j - \sum\nolimits_{i=1}^{k} \alpha_{ij} v_i \right\|_2$$

and then update the weights by setting

$$\forall i,\ 1 \le i \le k,\ \widetilde{w}_{ij} = w_{ij} + \sum\nolimits_{r=k+1}^{n_\ell} \alpha_{ir} w_{rj} \tag{4}$$

Using ordinary least squares to obtain α, the reweighting procedure runs in $\mathcal{O}(|\mathcal{T}|n_\ell^2 + n_\ell^3)$.

3 Experimental results

To quantify the performance of our algorithm, we present below the results of experiments[1] on common datasets for neural network evaluation: `MNIST` (LeCun and Cortes, 2010), `MNIST_ROT` (Larochelle et al., 2007) and `CIFAR-10` (Krizhevsky, 2009).

All networks were trained up until a certain training error threshold, using softmax activation on the output layer and sigmoids on other layers; see Table 1 for more details. In all following plots, error bars represent standard deviations.

[1]Run in MATLAB, based on the code from DeepLearnToolBox (`https://github.com/rasmusbergpalm/DeepLearnToolbox`) and Alex Kulesza's code for DPPs (`http://web.eecs.umich.edu/~kulesza/`), on a Linux Mint system with 16GB of RAM and an i7-4710HQ CPU @ 2.50GHz.

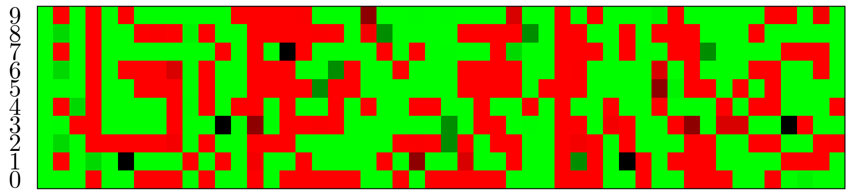
Table 1: Overview of the sets of networks used in the experiments. We train each class of networks until the first iteration of backprop for which the training error reaches a predefined threshold.

Dataset	Instances	Trained up until	Architecture
`MNIST`	5	$< 1\%$ error	784 - 500 - 500 - 10
`MNIST_ROT`	5	$< 1\%$ error	784 - 500 - 500 - 10
`CIFAR-10`	5	$< 50\%$ error	3072 - 1000 - 1000 - 1000 - 10

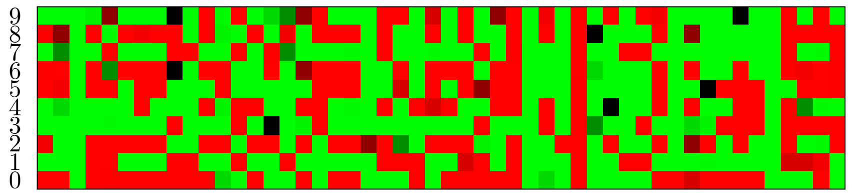
3.1 Pruning and reweighting analysis

To validate our claims on the benefits of using DPPs and fusing neurons, we compare these steps separately and also simultaneously against random pruning, where a fixed number of neurons are chosen uniformly at random from a layer and then removed, with and without our fusing step. We present performance results on the test data; of course, both DPP selection and reweighting are based solely on information drawn from the training data.

Figure 1 visualizes neuron activations in the first hidden layer of a network trained on the `MNIST` dataset. Each column in the plotted heat maps represents the activation of a neuron on instances of digits 0 through 9. Figure 1a shows the activations of the 50 neurons sampled using a k-DPP ($k = 50$) defined over the first hidden layer, whereas Figure 1b shows the activations of the first 50 neurons of the same layer. Figure 1b contains multiple similar columns: for example, there are 3 entirely green columns, corresponding to three neurons that saturate to 1 on each of the 10 instances. In contrast, the DPP samples neurons with diverse activations, and Figure 1a shows no similar redundancy.



(a) 50 neurons sampled via DPP from the first hidden layer



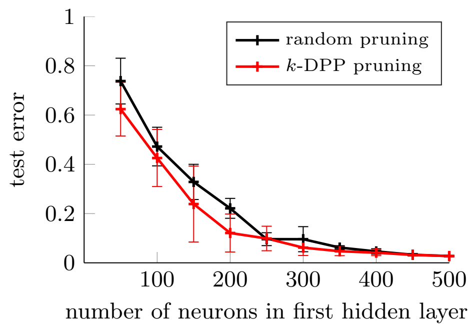
(b) First 50 neurons of the first hidden layer

Figure 1: Heat map of the activation of subsets of 50 neurons for one instance of each class of the `MNIST` dataset. The rows correspond to digits 0 through 9. Each column corresponds to the activation values of one neuron in the network's first layer on images of digits 0 through 9. Green values are activations close to 1, red values are activations close to 0.

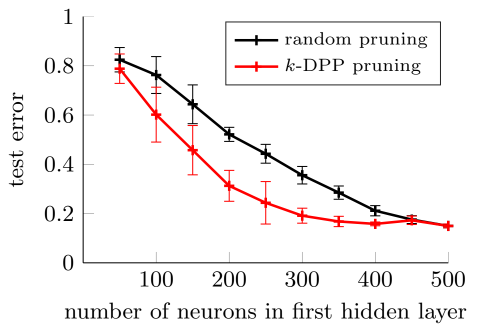
Figures 2 through 4 illustrate the impact of each step of DIVNET separately. Figure 2 shows the impact of pruning on test error using DPP pruning and random pruning (in which a fixed number of neurons are selected uniformly at random and removed from the network). DPP-pruned networks have consistently better training and test errors than networks pruned at random for the same final size. As expected, the more neurons are maintained, the less the error suffers from the pruning procedure; however, the pruning is in both cases destructive, and is seen to significantly increase the error rate.

This phenomenon can be mitigated by our reweighting procedure, as shown in Figure 3. By fusing and reweighting neurons after pruning, the training and test errors are considerably reduced, even when 90% of the layer's neurons are removed. Pruning also reduces variability of the results: the standard deviation for the results of the reweighted networks is significantly smaller than for the non-reweighted networks, and may be thus seen as a way to regularize neural networks.

Finally, Figure 4 illustrates the performance of DIVNET (DPP pruning and reweighting) compared to random pruning with reweighting. Although DIVNET's performance is ultimately better, the reweighting procedure also dramatically benefits the networks that were pruned randomly.

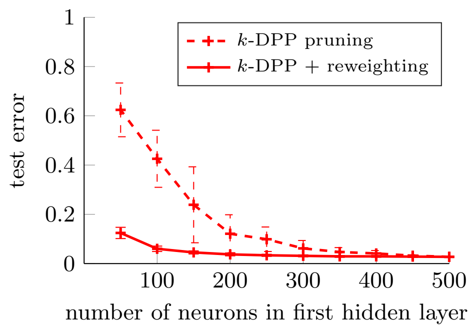


(a) `MNIST` dataset

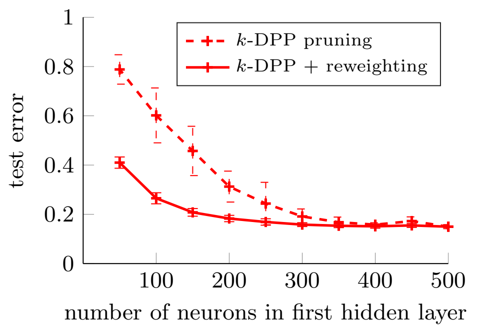


(b) `MNIST_ROT` dataset

Figure 2: Comparison of random and k-DPP pruning procedures.

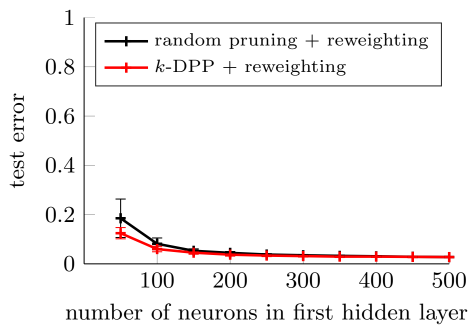


(c) `MNIST` dataset

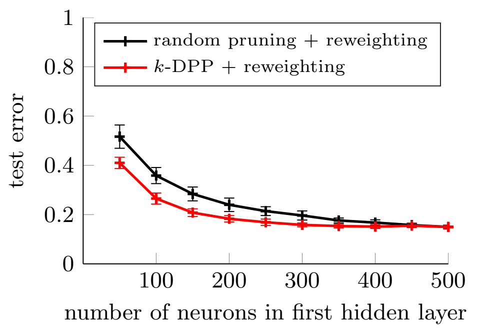


(d) `MNIST_ROT` dataset

Figure 3: Comparison of DIVNET (k-DPP + reweighting) to simple k-DPP pruning.



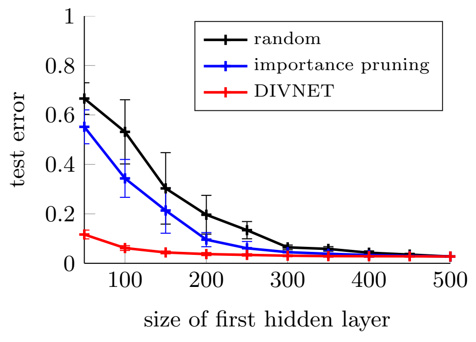
(a) `MNIST` dataset



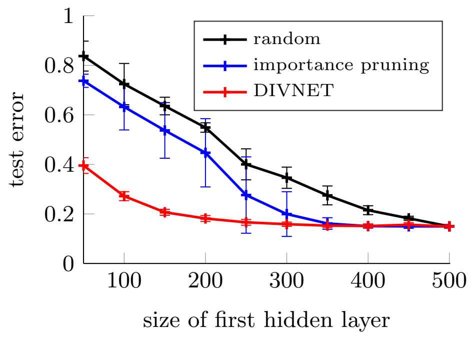
(b) `MNIST_ROT` dataset

Figure 4: Comparison of random and k-DPP pruning when both are followed by reweighting.

We also ran these experiments on networks for shrinking the second layer while maintaining the first layer intact. The results are similar, and may be found in Appendix A. Notably, we found that the gap between DIVNET and random pruning's performances was much wider when pruning the last layer. We believe this is due to the connections to the output layer



(a) MNIST dataset



(b) MNIST_ROT dataset

Figure 5: Comparison of random pruning, importance pruning, and DIVNET's impact on the network's performance after decreasing the number of neurons in the first hidden layer of a network.

being learned much faster, thus letting a small, diverse subset of neurons (hence well suited to DPP sampling) in the last hidden layer take over the majority of the computational effort.

3.2 Performance analysis

Much attention has been given to reducing the size of neural networks in order to reduce memory consumption. When using neural nets locally on devices with limited memory, it is crucial that their memory footprint be as small as possible.

Node importance-based pruning (henceforth "importance pruning") is one of the most intuitive ways to cut down on network size. Introduced to deep networks by He et al. (2014), this method removes the neurons whose calculations impact the network the least. Among the three solutions to estimating a neuron's importance discussed in He et al. (2014), the sum the output weights of each neuron (the 'onorm' function) provided the best results:

$$\text{onorm}(n_i) := \frac{1}{n_{\ell+1}} \sum\nolimits_{j=1}^{n_\ell} |w_{ij}^{\ell+1}|.$$

Figure 5 compares the test data error of the networks after being pruned using importance pruning that uses onorm as a measure of relevance against DIVNET. Since importance pruning deletes neurons that contribute the least to the next layer's computations, it performs well up to a certain point; however, when pruning a significant amount of neurons, this pruning procedure even removes neurons performing essential calculations, hurting the network's performance significantly. However, since DIVNET fuses redundant neurons, instead of merely deleting them its resulting network delivers much better performance even with very large amounts of pruning.

In order to illustrate numerically the impact of DIVNET on network performance, Table 2 shows network training and test errors (between 0 and 1) under various compression rates obtained with DIVNET, without additional retraining (that is, the pruned network is not retrained to re-optimize its weights).

Table 2: Training and test error for different percentages of remaining neurons (mean ± standard deviation). Initially, MNIST and MNIST_ROT nets have 1000 hidden neurons, and CIFAR-10 have 3000.

Remaining hidden neurons		10%	25%	50%	75%	100 %
MNIST	training error	0.76 ± 0.06	0.28 ± 0.12	0.15 ± 0.04	0.06 ± 0.04	0.01 ±0.001
	test error	0.76 ± 0.07	0.29 ± 0.12	0.17 ± 0.05	0.07 ± 0.03	0.03 ± 0.002
MNIST_ROT	training error	0.74 ± 0.08	0.54 ± 0.09	0.34 ± 0.06	0.20 ± 0.03	0.01 ± 0.003
	test error	0.73 ± 0.09	0.49 ± 0.11	0.25 ± 0.07	0.06 ± 0.03	0.15 ±0.008
CIFAR-10	training error	0.84 ± 0.05	0.61 ± 0.01	0.52 ± 0.01	0.50 ± 0.01	0.49 ± 0.004
	test error	0.85 ± 0.05	0.62 ± 0.02	0.54 ± 0.01	0.52 ± 0.01	0.51 ± 0.005

3.3 Discussion and Remarks

- In all experiments, sampling and reweighting ran several orders of magnitude faster than training; reweighting required significantly more time than sampling. If Divnet must be further sped up, a fraction of the training set can be used instead of the entire set, at the possible cost of subsequent network performance.
- When using DPPs with a Gaussian RBF kernel, sampled neurons need not have linearly independent activation vectors: not only is the DPP sampling probabilistic, the kernel itself is not scale invariant. Indeed, for two collinear but unequal activation vectors, the corresponding coefficient in the kernel will not be 1 (or γ with the $L \leftarrow \gamma L$ update).
- In our work, we selected a subset of neurons by sampling once from the DPP. Alternatively, one could sample a fixed amount of times, using the subset with the highest likelihood (i.e., largest $\det(L_Y)$), or greedily approximate the mode of the DPP distribution.
- Our reweighting procedure benefits competing pruning methods as well (see Figure 4).
- We also investigated DPP sampling for pruning concurrently with training iterations, hoping that this might allow us to detect superfluous neurons before convergence, and thus reduce training time. However, we observed that in this case DPP pruning, with or without reweighting, did not offer a significant advantage over random pruning.
- Consistently over all datasets and networks, the expected sample size from the kernel L' was much smaller for the last hidden layer than for other layers. We hypothesize that this is caused by the connections to the output layer being learned faster than the others, allowing a small subset of neurons to take over the majority of the computational effort.

4 Future work and conclusion

Divnet leverages similarities between the behaviors of neurons in a layer to detect redundant parameters and merge them, thereby enforcing neuronal diversity within each hidden layer. Using Divnet, large, redundant networks can be shrunk to much smaller structures without impacting their performance and without requiring further training. We believe that the performance profile of Divnet will remain similar even when scaling to larger scale datasets and networks, and hope to include results on bigger problems (e.g., Imagenet (Russakovsky et al., 2015)) in the future.

Many hyper-parameters can be tuned by a user as per need include: the number of remaining neurons per layer can be fixed manually; the precision of the reweighting and the sampling procedure can be tuned by choosing how many training instances are used to generate the DPP kernel and the reweighting coefficients, creating a trade-off between accuracy, memory management, and computational time. Although Divnet requires the user to select the size of the final network, we believe that a method where no parameter explicitly needs to be tuned is worth investigating. The fact that DPPs can be augmented to also reflect different distributions over the sampled set sizes (Kulesza and Taskar, 2012, §5.1.1) might be leveraged to remove the burden of choosing the layer's size from the user.

Importantly, Divnet is agnostic to most parameters of the network, as it only requires knowledge of the activation vectors. Consequently, Divnet can be easily used jointly with other pruning/memory management methods to reduce size. Further, the reweighting procedure is agnostic to how the pruning is done, as shown in our experiments.

Furthermore, the principles behind Divnet can theoretically also be leveraged in non fully-connected settings. For example, the same diversifying approach may also be applicable to filters in CNNs: if a layer of the CNN is connected to a simple, feed-forward layer – such as the S4 layer in LeCun et al. (1998) – by viewing each filter's activation values as an vector and applying Divnet on the resulting activation matrix, one may be able to remove entire filters from the network, thus significantly reducing CNN's memory footprint.

Finally, we believe that investigating DPP pruning with different kernels, such as kernels invariant to the scaling of the activation vectors, or even kernels learned from data, may provide insight into which interactions between neurons of a layer contain the information necessary for obtaining good representations and accurate classification. This marks an interesting line of future investigation, both for training and representation learning.

Acknowledgments

This work is partly supported by NSF grant: IIS-1409802.

References

W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *CoRR*, abs/1504.04788, 2015.

M. Courbariaux, Y. Bengio, and J. David. Low precision arithmetic for deep learning. *CoRR*, abs/1412.7024, 2014.

M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning. *CoRR*, abs/1306.0543, 2013.

S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. *CoRR*, abs/1502.02551, 2015.

S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

B. Hassibi, D. G. Stork, and S. C. R. Com. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*, pages 164–171. Morgan Kaufmann, 1993.

T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu. Reshaping deep neural network for fast decoding by node-pruning. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 245–249, May 2014.

G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

J. B. Hough, M. Krishnapur, Y. Peres, and B. Virág. Determinantal processes and independence. *Probability Surveys*, 3(206–229):9, 2006.

B. Kang. Fast determinantal point process sampling with application to clustering. In *Advances in Neural Information Processing Systems 26*, pages 2319–2327. Curran Associates, Inc., 2013.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

A. Kulesza and B. Taskar. k-DPPs: Fixed-size determinantal point processes. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

A. Kulesza and B. Taskar. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5(2–3), 2012.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 473–480, 2007.

Y. LeCun and C. Cortes. MNIST handwritten digit database, 2010. URL `http://yann.lecun.com/exdb/mnist/`.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

C. Li, S. Jegelka, and S. Sra. Efficient sampling for k-determinantal point processes. *preprint*, 2015. URL `http://arxiv.org/abs/1509.01618`.

O. Macchi. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, 7(1), 1975.

D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, July 2015.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

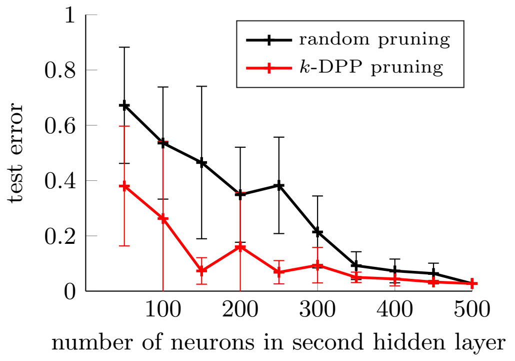
T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013*, pages 6655–6659. IEEE, 2013.

J. Snoek, R. Zemel, and R. P. Adams. A determinantal point process latent variable model for inhibition in neural spiking data. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1932–1940. Curran Associates, Inc., 2013.

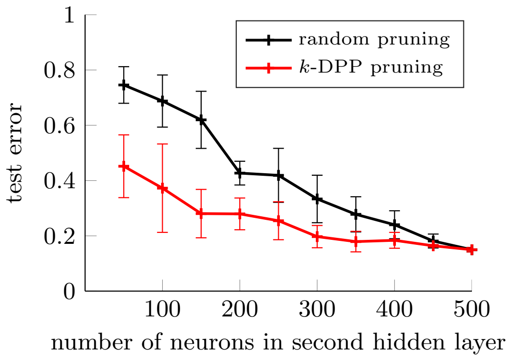
S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *CoRR*, abs/1507.06149, 2015. URL `http://arxiv.org/abs/1507.06149`.

Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. J. Smola, L. Song, and Z. Wang. Deep fried convnets. *CoRR*, abs/1412.7149, 2014.

A Pruning the second layer

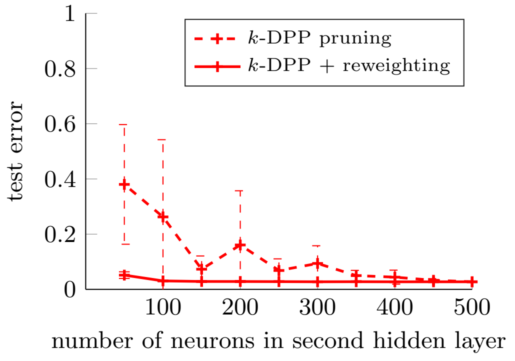


(a) `MNIST` dataset

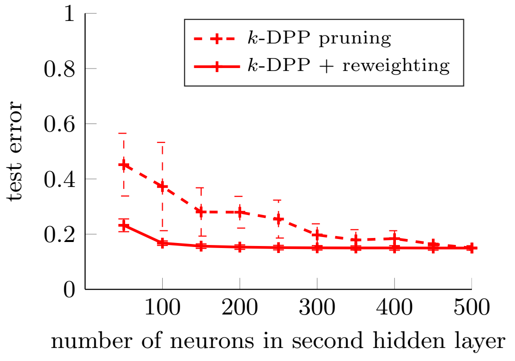


(b) `MNIST_ROT` dataset

Figure 6: Comparison of random and k-DPP pruning procedures.

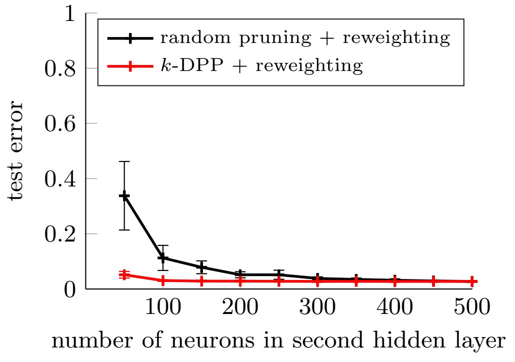


(a) `MNIST` dataset

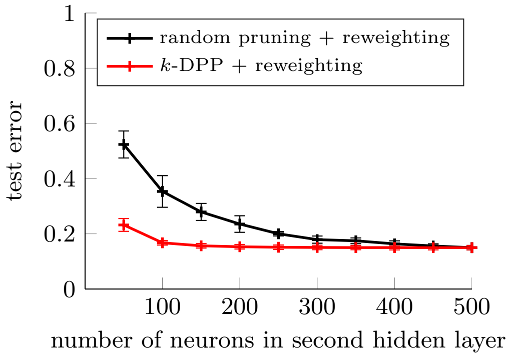


(b) `MNIST_ROT` dataset

Figure 7: Comparison of DivNet to simple k-DPP pruning.



(a) `MNIST` dataset



(b) `MNIST_ROT` dataset

Figure 8: Comparison of random and k-DPP pruning when both are followed by reweighting.

B Influence of the β parameter on network size and error

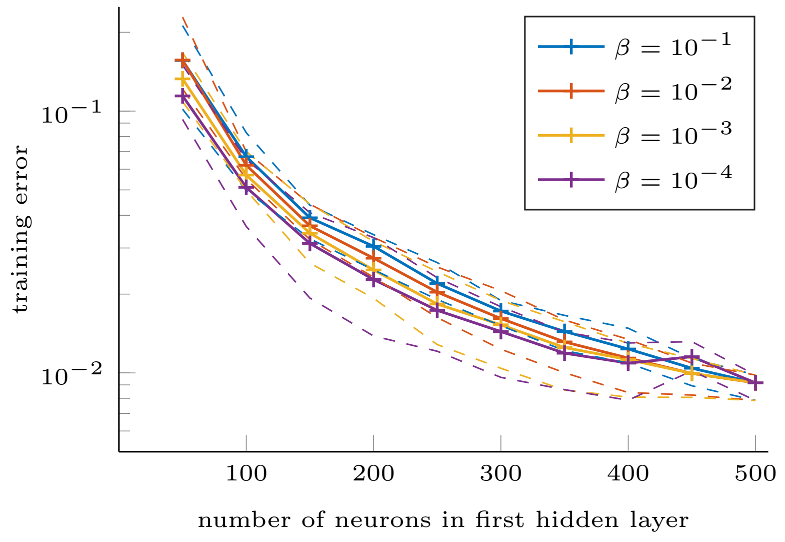


Figure 9: Influence of β on training error (using the networks trained on `MNIST`). The dotted lines show min and max errors.

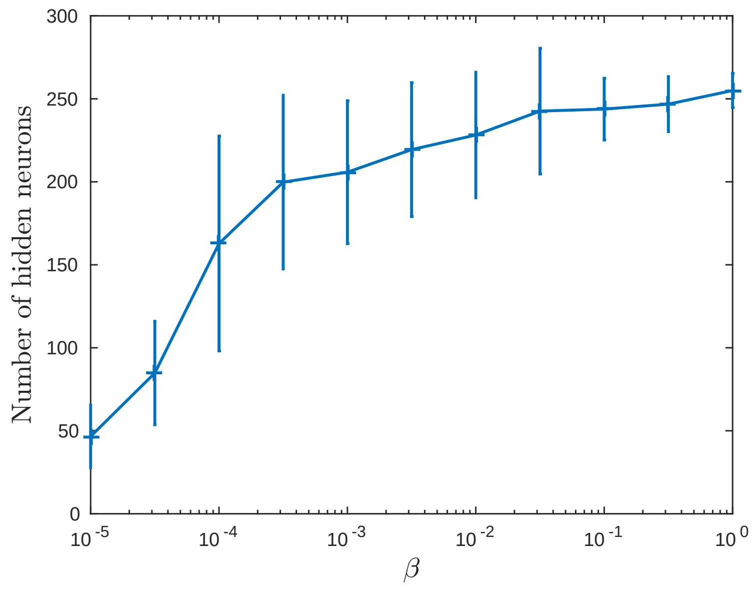
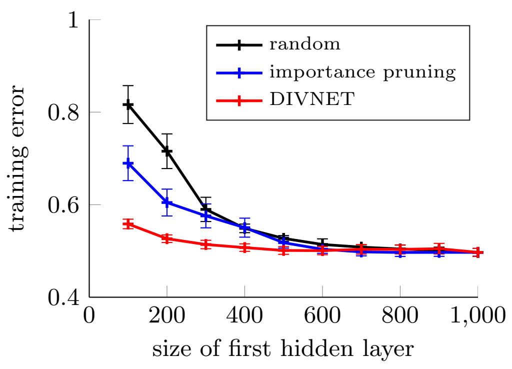
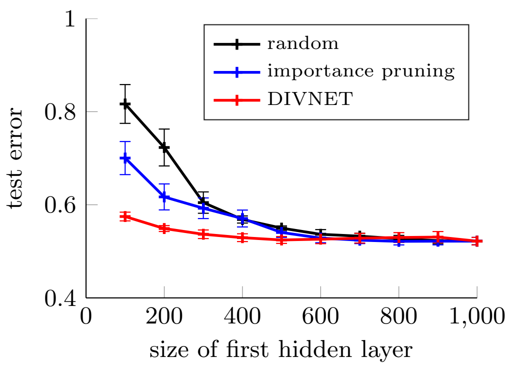


Figure 10: Influence of β on the number of neurons that remain after pruning networks trained on `MNIST` (when pruning non-parametrically, using a DPP instead of a k-DPP.)

C Comparison of Divnet to importance-based pruning and random pruning on the CIFAR-10 dataset



(a) Training error on `CIFAR-10` dataset



(b) Test error on `CIFAR-10` dataset

Figure 11: Comparison of random pruning, importance pruning, and Divnet's impact on the network's performance after decreasing the number of parameters in the network.