Targeted Dropout

Aidan N. Gomez
Google Brain
aidan.gomez@cs.ox.ac.uk

Ivan Zhang
FOR.ai
ivan@for.ai

Kevin Swersky
Google Brain
kswersky@google.com

Yarin Gal
University of Oxford
yarin@cs.ox.ac.uk

Geoffrey E. Hinton
Google Brain
geoffhinton@google.com

Abstract

Neural networks are extremely flexible models due to their large number of parameters, which is beneficial for learning, but also highly redundant. This makes it possible to compress neural networks without having a drastic effect on performance. We introduce targeted dropout, a strategy for post hoc pruning of neural network weights and units that builds the pruning mechanism directly into learning. At each weight update, targeted dropout selects a candidate set for pruning using a simple selection criterion, and then stochastically prunes the network via dropout applied to this set. The resulting network learns to be explicitly robust to pruning, comparing favourably to more complicated regularization schemes while at the same time being extremely simple to implement, and easy to tune.

1 Introduction

There has been a great deal of work on developing strategies to sparsify neural networks [10, 7, 5, 2, 13, 3, 12, 9, 18]. Sparsification involves removing weights (corresponding to setting them to 0) or entire units from the network, while maintaining predictive performance. Sparsity can be encouraged during learning by the use of sparsity-inducing regularizers, like L^1 or L^0 penalties. It can also be imposed by *post hoc* pruning, where a full-sized network is trained, and then sparsified according to some pruning strategy. Ideally, given some measurement of task performance, we would prune the weights or units that provide the least amount of benefit to the task. Finding the optimal set is in general a difficult combinatorial problem, and even a greedy strategy would require an unrealistic number of task evaluations, as there are often millions of parameters in a modern neural network architecture. Common pruning strategies therefore focus on fast approximations, such as removing weights with the smallest magnitude [6], or ranking the weights by the sensitivity of the task performance with respect to the weights and removing the least-sensitive ones [10].

Our approach is based on the observation that dropout regularization [8, 17] itself enforces sparsity during training, by sparsifying the network with each forward pass. This encourages the network to learn a representation that is robust to a particular form of post hoc sparsification – in this case, where a random set of units are removed. Our hypothesis is that if we plan to do explicit post hoc sparsification, then we can do better by specifically applying dropout to the set of units that we a priori believe are the least useful. We call this approach *targeted dropout*. The idea is to rank weights or units according to some fast, approximate measure of importance (like magnitude), and then apply dropout primarily to those elements with *low importance*. Similar to the observation with regular dropout, we show that this encourages the network to learn a representation where the importance of weights or units more closely aligns with our approximation. In other words, the

Code available at: `github.com/for-ai/TD`

CDNNRIA Workshop at the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

network learns to be robust to *our choice* of post hoc pruning strategy. The advantage of targeted dropout compared to other approaches is that it leads to a converged network that is extremely robust to post hoc pruning. It is simultaneously easy to implement, consisting of a two-line change using neural network frameworks such as Tensorflow [1] or PyTorch [14]. Furthermore, it is explicit: the desired level of sparsity is provided by the user, and enforced throughout training.

2 Targeted Dropout

2.1 Dropout

Our work uses the two most popular Bernoulli dropout techniques, Hinton et al.'s unit dropout [8, 17] and Wan et al.'s weight dropout (dropconnect) [20]. For a fully-connected layer with input tensor $\boldsymbol{X}$, weight matrix $\mathbf{W}$, output tensor $\boldsymbol{Y}$, and mask $\mathbf{M}_{i,o} \sim \mathrm{Bernoulli}(\alpha)$ we define both techniques below:

Unit dropout [8, 17]:

$$\boldsymbol{Y} = (\boldsymbol{X} \odot \mathbf{M})\mathbf{W}$$

Unit dropout randomly drops *units* (sometimes referred to as neurons) at each training step to reduce dependence between units and prevent overfitting.

Weight dropout [20]:

$$\boldsymbol{Y} = \boldsymbol{X}(\mathbf{W} \odot \mathbf{M})$$

Weight dropout randomly drops individual *weights* in the weight matrices at each training step. Intuitively, this is dropping connections between layers, forcing the network to adapt to a different connectivity at each training step.

2.2 Magnitude-based pruning

A popular class of pruning strategies are those characterized as magnitude-based pruning strategies. These strategies treat the top-k largest magnitude weights as important. We use $\mathrm{argmax\text{-}k}$ to return the top-k elements (units or weights) out of all elements being considered.

Unit pruning [6]: considers the units (column-vectors) of weight matrices under the L^2-norm.

$$\mathcal{W}(\boldsymbol{\theta}) = \left\{ \underset{1 \leq o \leq N_{\mathrm{col}}(\mathbf{W})}{\mathrm{argmax\text{-}k}} \; \|\boldsymbol{w}_o\|_2 \;\middle|\; \mathbf{W} \in \boldsymbol{\theta} \right\} \tag{1}$$

Weight pruning [10]: considers each entry of the weight matrix separately under the L^1-norm. Note that the top-k is with respect to the other weights in the same filter.

$$\mathcal{W}(\boldsymbol{\theta}) = \left\{ \underset{1 \leq i \leq N_{\mathrm{row}}(\mathbf{W})}{\mathrm{argmax\text{-}k}} \; |\mathbf{W}_{io}| \;\middle|\; 1 \leq o \leq N_{\mathrm{col}}(\mathbf{W}), \mathbf{W} \in \boldsymbol{\theta} \right\} \tag{2}$$

While weight pruning tends to preserve more of the task performance under coarser prunings [5, 19, 4], unit pruning allows for considerably greater computational savings [21, 12].

2.3 Methods

Consider a neural network parameterized by $\boldsymbol{\theta}$, and our pruning strategy (defined above in Equations (1) and (2)) $\mathcal{W}$. We hope to find optimal parameters $\boldsymbol{\theta}^*$ such that our loss $\mathcal{E}(\mathcal{W}(\boldsymbol{\theta}^*))$ is low and at the same time $|\mathcal{W}(\boldsymbol{\theta}^*)| \leq k$, i.e. we wish to keep only the k weights of highest magnitude in the network. A deterministic implementation would select the bottom $|\boldsymbol{\theta}| - k$ elements and drop them out. However, we would like for low-valued elements to be able to increase their value if they become important during training. Therefore, we introduce stochasticity into the process using a targeting proportion γ and a drop probability α. The targeting proportion means that we select the bottom $\gamma|\boldsymbol{\theta}|$ weights as candidates for dropout, and of those we drop the elements independently with drop rate α. This implies that the expected number of units to keep during each round of targeted dropout is $(1 - \gamma \cdot \alpha)|\boldsymbol{\theta}|$. As we will see below, the result is a reduction in the important subnetwork dependency on the unimportant subnetwork, thereby reducing the performance degradation of pruning at the conclusion of training.

Table 1: ResNet-32 model accuracies on CIFAR-10 at differing pruning percentages and under different regularization schemes. The top table depicts results using the weight pruning strategy, while the bottom table depicts the results of unit pruning (see Sec. 2.2).

Weight Dropout/Pruning

prune percentage	none	dropout $\alpha{=}0.25$	targeted $\alpha{=}0.33,\gamma{=}0.75$	targeted $\alpha{=}0.5,\gamma{=}0.75$	targeted $\alpha{=}0.66,\gamma{=}0.75$	targeted $\alpha{=}0.5,\gamma{=}0.5$	targeted $\alpha{=}0.5,\gamma{=}0.5$ $+ L^1_{0.1}$	variational	$L^1_{0.1}$
0 %	94.30	94.11	90.45	91.67	93.87	94.38	93.90	92.75	93.95
10%	94.16	94.05	90.42	91.71	93.85	94.31	93.95	92.83	93.84
20%	94.19	93.97	90.43	91.72	93.84	94.27	93.89	92.67	93.88
30%	94.21	93.90	90.42	91.73	93.89	94.27	93.89	92.52	93.80
40%	93.93	93.59	90.38	91.71	93.81	94.27	93.91	92.12	93.81
50%	93.31	92.00	90.38	91.76	93.84	94.33	93.94	90.98	93.47
60%	91.50	88.23	90.41	91.74	93.89	93.92	93.66	80.91	92.02
70%	82.89	58.07	90.40	91.68	93.84	92.19	92.80	44.32	88.07
80%	38.35	10.66	85.10	89.63	92.31	74.31	87.38	11.30	58.97
90%	12.76	9.92	12.76	41.60	46.57	16.67	28.43	10.02	15.13

Unit Dropout/Pruning

prune percentage	none	dropout $\alpha{=}0.25$	targeted $\alpha{=}0.33,\gamma{=}0.75$	targeted $\alpha{=}0.5,\gamma{=}0.75$	targeted $\alpha{=}0.66,\gamma{=}0.75$	targeted $\alpha{=}0.5,\gamma{=}0.5$	targeted $\alpha{=}0.5,\gamma{=}0.5$ $+ L^1_{0.1}$	variational	$L^1_{0.01}$
0 %	94.29	92.93	91.00	90.15	90.55	93.27	93.08	92.73	94.68
10%	90.38	84.98	85.92	88.89	90.83	92.87	92.90	92.73	91.38
20%	74.38	21.27	61.78	74.64	89.88	91.95	92.72	92.74	81.10
30%	36.49	10.32	19.21	59.02	87.35	89.84	91.97	92.63	30.55
40%	12.64	12.52	14.32	27.17	85.39	86.41	90.55	91.90	21.88
50%	10.19	10.13	11.95	15.97	80.84	67.01	83.23	90.16	13.12
60%	11.32	9.95	11.16	12.67	71.97	11.70	34.13	79.03	9.95
70%	10.14	9.98	10.03	10.72	55.98	9.98	12.14	23.13	9.70
80%	10.01	9.91	09.94	09.92	10.02	9.94	10.07	10.20	10.69
90%	10.12	9.95	09.86	09.95	10.07	9.95	9.90	10.05	9.88

Table 2: Comparing Smallify to targeted dropout and ramping targeted dropout. Experiments on CIFAR10 using ResNet32. Left: the best of the three targeted dropout runs compared against the best out of six smallify runs; Middle: inspecting higher pruning rates of the best smallify run compared to ramping targeted dropout; Right: inspecting even higher pruning rates of ramping targeted dropouts.

Weight Dropout/Pruning

prune percentage	targeted $\alpha{=}0.66,\gamma{=}0.75$	smallify1en51
0 %	93.87	91.50
10%	93.85	91.51
20%	93.84	91.48
30%	93.89	91.52
40%	93.81	91.53
50%	93.84	91.51
60%	93.89	91.55
70%	93.84	91.57
80%	92.31	91.55
90%	46.57	91.45

	smallify $\lambda{=}0.00001$	ramp targ $\alpha{=}0.99,\gamma{=}0.99$
90%	91.48	88.70
91%	91.34	88.74
92%	91.33	88.75
93%	91.41	88.75
94%	90.80	88.80
95%	90.30	88.73
96%	87.54	88.74
97%	70.29	88.67
98%	33.04	88.70
99%	10.46	88.75

	ramp targ $\alpha{=}0.99,\gamma{=}0.99$
98.5%	88.74
98.6%	88.82
98.7%	88.79
98.8%	88.74
98.9%	88.70
99.0%	88.71
99.1%	87.63
99.2%	67.25
99.3%	51.86
99.4%	11.59

Our weight pruning experiments (shown in Tables 1 & 2) demonstrate that the baseline regularization schemes are comparatively weak to their targeted counterparts. We find that targeted dropout applied to the weights results in the network outperforming unregularized performance with only half the total number of parameters. We are able to reach extremely high rates of pruning in Table 2 by gradually annealing the targeted proportion from zero percent of weights up to ninety-nine percent throughout the course of training.

3 Conclusion

We propose *targeted dropout* as a simple and extremely effective regularization tool for incorporating post hoc pruning strategies into the training procedure of neural networks without drastic impact to underlying task performance of a particular architecture. Among the primary benefits of targeted dropout are the simplicity of implementation and intuitive, flexible hyperparameters.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015. URL `http://download.tensorflow.org/paper/whitepaper2015.pdf`.

[2] Mohammad Babaeizadeh, Paris Smaragdis, and Roy H Campbell. Noiseout: A simple way to prune neural networks. *arXiv preprint arXiv:1611.06211*, 2016.

[3] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4860–4874, 2017.

[4] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[5] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[6] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[7] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[8] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[9] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. *arXiv preprint arXiv:1801.07365*, 2018.

[10] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[13] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. 2016.

[14] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[15] Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pages 1746–1754, 2014.

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

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

[18] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

[19] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

[20] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[21] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

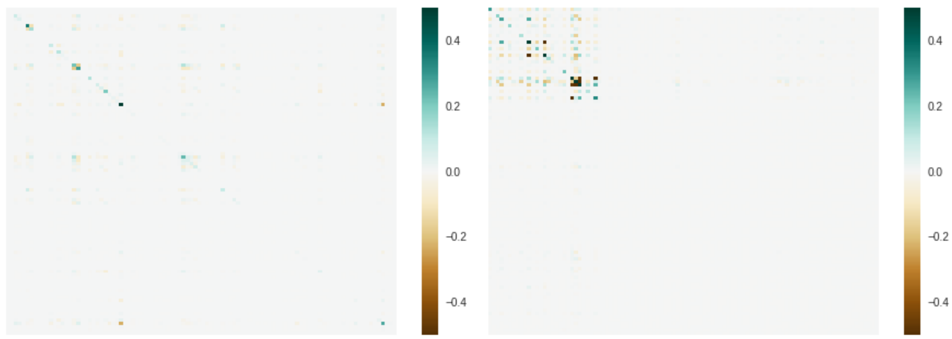

Figure 1: A comparison between a network without dropout (left) and with targeted dropout (right) of the matrix formed by $\theta^\top \odot \boldsymbol{H} \odot \theta$. The weights are ordered such that the last 75% are the weights with the lowest magnitude (those we intend to prune). The sum of the elements of the lower right hand corner – formed by the intersection of the elements in the rightmost 3/4 of columns with the lowermost 3/4 of rows – approximates the change in error after pruning (Eqn. (3)). Note the stark difference between the two networks, with targeted dropout concentrating its dependence on the top left corner, leading to a much smaller error change after pruning.

Appendix

Dependence Between the Important and Unimportant Subnetworks

The goal of targeted dropout is to reduce the dependence of the important subnetwork on its complement. A commonly used intuition behind dropout is the prevention of coadaptation between units; that is, when dropout is applied to a unit, the remaining network can no longer depend on that unit's contribution to the function and must learn to propagate that unit's information through a more reliable channel. An alternative description asserts that dropout maximizes the mutual information between units in the same layer, thereby decreasing the impact of losing a unit [17]. Dropout has also been shown to impose 'hierarchy' among units depending on the particular drop rate associated with each unit [15].

A more relevant intuition into the effect of targeted dropout in our specific pruning scenario can be obtained from an illustrative case where the important subnetwork is completely separated from the unimportant one. Suppose a network was composed of two non-overlapping subnetworks, each able to produce the correct output by itself, with the network output given as the average of both subnetwork outputs. If our importance criterion designated the first subnetwork as important, and the second subnetwork as unimportant (more specifically, it has lower weight magnitude), then adding noise to the weights of the unimportant subnetwork (i.e. applying dropout) means that with non-zero probability we will corrupt the network output. Since the important subnetwork is already able to predict the output correctly, to reduce the loss we must therefore reduce the weight magnitude of the unimportant subnetwork output layer towards zero, in effect "killing" that subnetwork, and reinforcing the separation between the important subnetwork and the unimportant one.

These interpretations make clear why dropout should be considered a natural tool for application in pruning. We can empirically confirm targeted dropout's effect on weight dependence by comparing a network trained with and without targeted dropout and inspecting the Hessian and gradient to determine the dependence of the network on the losing ticket (i.e. our weights to be pruned). As in LeCun et al. [10], we can estimate the effect of pruning weights by considering the second degree Taylor expansion of change in loss:

$$|\mathcal{E}(\boldsymbol{\theta} - \boldsymbol{d}) - \mathcal{E}(\boldsymbol{\theta})| = | - \nabla_{\boldsymbol{\theta}} \mathcal{E}^\top \boldsymbol{d} + {}^1\!/\!{}_2\, \boldsymbol{d}^\top \boldsymbol{H} \boldsymbol{d} + \mathcal{O}(\|\boldsymbol{d}\|^3)| \tag{3}$$

Where $d_i = \theta_i$ if $\theta_i \in \overline{\mathcal{W}(\boldsymbol{\theta})}$ and 0 otherwise. $\nabla_{\boldsymbol{\theta}} \mathcal{E}$ are the gradients of the loss, and $\boldsymbol{H}$ is the Hessian. Note that at the end of training, if we have found a critical point $\boldsymbol{\theta}^*$, then $\nabla_{\boldsymbol{\theta}} \mathcal{E}(\boldsymbol{\theta}^*) = \mathbf{0}$, leaving only the Hessian term. In our experiments we empirically confirm that targeted dropout reduces the dependence between the important and unimportant subnetworks by an order of magnitude (See Fig. 1).

Table 3: LeNet-5 model accuracies on CIFAR-10 at differing pruning percentages and under different regularization schemes. Table top table depicts results using the weight pruning strategy, while bottom table depicts the results of unit pruning (see Sec. 2.2).

Weight Dropout/Pruning

prune percentage	none	dropout $\alpha{=}0.25$	targeted $\alpha{=}0.5,\gamma{=}0.5$	targeted $\alpha{=}0.33,\gamma{=}0.75$	targeted $\alpha{=}0.66,\gamma{=}0.75$	$L^1_{0.1}$	variational	$L^0_{0.1}$	$L^0_{0.01}$	$L^0_{0.001}$	$L^0_{0.0001}$
0%	67.69	59.84	66.51	65.98	65.16	69.91	66.76	65.46	61.77	66.88	65.84
10%	67.10	59.82	66.55	65.88	65.15	69.57	66.76	65.39	61.83	66.80	65.86
20%	68.15	59.76	66.66	66.22	65.16	69.45	66.76	65.32	61.91	66.98	65.96
30%	66.79	59.26	66.59	66.26	65.48	67.47	66.76	65.70	61.94	66.81	65.66
40%	63.23	58.91	66.96	66.04	65.70	61.44	66.67	65.33	62.34	65.82	65.55
50%	50.16	58.98	67.38	66.34	66.00	47.63	66.75	65.30	61.82	64.97	64.59
60%	35.44	56.38	65.52	66.79	66.30	35.47	66.82	63.77	60.73	64.55	61.92
70%	33.53	53.13	53.69	66.77	66.63	23.55	49.62	59.80	58.12	62.39	55.88
80%	26.78	42.56	29.23	55.70	58.23	15.62	14.69	46.77	51.80	53.78	47.19
90%	24.24	24.71	16.63	31.11	25.09	17.25	16.33	31.79	39.33	45.14	37.59

Unit Dropout/Pruning

prune percentage	none	dropout $\alpha{=}0.25$	targeted $\alpha{=}0.5,\gamma{=}0.5$	targeted $\alpha{=}0.33,\gamma{=}0.75$	targeted $\alpha{=}0.66,\gamma{=}0.75$	$L^1_{0.1}$	variational	$L^0_{0.1}$	$L^0_{0.01}$	$L^0_{0.001}$	$L^0_{0.0001}$
0%	67.69	61.02	64.31	62.05	56.11	69.91	69.27	70.30	66.54	68.49	69.14
10%	53.89	60.07	64.09	61.31	55.95	59.39	69.20	60.99	59.59	65.68	61.13
20%	34.72	56.67	63.30	60.38	55.11	33.86	69.43	27.52	34.43	51.54	46.61
30%	31.86	55.19	63.25	58.98	54.92	31.98	68.94	27.50	31.51	49.04	43.50
40%	22.52	49.13	60.23	55.09	53.41	22.05	66.59	23.42	21.48	28.91	27.48
50%	11.57	40.11	56.27	51.41	51.52	17.15	51.96	13.82	16.45	21.27	18.94
60%	11.53	33.80	41.57	49.39	50.11	17.23	19.43	13.37	16.42	20.62	20.38
70%	11.06	15.47	16.96	41.01	46.13	13.48	10.05	13.84	13.34	11.28	13.67
80%	11.04	13.36	15.02	38.61	44.15	13.57	9.99	12.19	14.16	12.03	11.48
90%	10.61	11.05	10.31	12.15	13.46	11.24	10.05	9.85	10.58	8.34	8.75

LeNet-5

Our first suite of experiments are performed on the LeNet-5 architecture [11] applied to the CIFAR-10 dataset. This architecture is used as a control as it is comparatively shallow and underperforming relative to the ResNet architecture used in the next section. The network is trained for 256 epochs by SGD with momentum of 0.9 and a constant learning rate of 0.01. We apply basic input augmentation in the form of random crops, random horizontal flipping, and standardization.

In Table 3 we compare both weight and unit targeted dropout against standard weight and unit dropout, as well as L^1 regularization, variational dropout, and L^0 regularization. We find that both forms of targeted dropout outperform their standard dropout counterpart; we also find that the version of targeted dropout used (unit versus weight) must match the version of pruning strategy used post hoc. Variational dropout performs fairly well in both its weight and unit forms, however the sparsity is effectively fixed at the rate discovered by the optimization procedure, and the model accuracy breaks down rapidly once this threshold is surpassed. The weight-level variant of L^0 regularization performs very well on LeNet and is capable of sustaining significant pruning; however, targeted dropout is superior for all rates over 50% aside from 90%. The unit-level variant performs only marginally better than the unregularized baseline, and is significantly outperformed by targeted dropout. Regarding L^1 regularization, we find that it doesn't seem to improve over the unregularized baseline.

VGG-16

In order to experiment with a different architecture/data pair we experiment with the VGG-16 architecture [16] on CIFAR-100. We find that targeted dropout behaves similarly across all dataset-architecture pairs (see Table 4).

Table 4: VGG-16 applied to CIFAR-100.

Weight Dropout/Pruning

prune percentage	none	dropout $\alpha=0.25$	targeted $\alpha=0.5,\gamma=0.5$	targeted $\alpha=0.66,\gamma=0.75$	variational	$L^0_{0.001}$	smallify $\lambda=0.00001$
0 %	59.59	59.84	58.71	58.35	49.45	60.05	55.97
10%	59.16	59.73	58.67	58.34	49.40	59.94	55.89
20%	57.94	59.48	58.67	58.30	47.81	59.75	55.85
30%	53.66	59.50	58.68	58.22	46.64	59.80	55.76
40%	44.01	58.87	58.72	58.39	42.18	58.01	54.60
50%	30.92	55.34	58.60	58.38	41.97	54.13	53.29
60%	20.77	42.09	49.04	58.27	34.55	41.40	46.49
70%	15.37	17.34	23.55	58.24	22.38	20.07	45.16
80%	3.82	2.48	4.39	51.71	13.66	4.73	29.60
90%	0.96	1.05	1.41	3.70	4.86	1.02	11.73

Unit Dropout/Pruning

prune percentage	none	dropout $\alpha=0.25$	targeted $\alpha=0.5,\gamma=0.5$	variational	$L^0_{0.001}$	smallify $\lambda=0.0001$
0 %	59.46	45.97	52.09	41.97	63.77	59.31
10%	50.27	34.46	52.48	42.14	19.45	58.15
20%	8.77	2.22	52.98	41.94	1.18	39.66
30%	2.07	1.02	52.24	39.46	1.02	12.34
40%	0.99	1.05	51.62	37.87	1.00	2.86
50%	1.58	1.04	48.35	31.77	1.02	0.98
60%	1.01	1.04	1.44	17.20	0.96	0.95
70%	0.95	0.96	1.05	13.13	0.92	0.99
80%	0.96	0.97	0.94	8.33	0.95	0.95
90%	1.02	0.98	0.97	4.57	0.94	1.01