Training Thinner and Deeper Neural Networks: Jumpstart Regularization

Carles Riera[1], Camilo Rey[1], Thiago Serra[2], Eloi Puertas[1], and Oriol Pujol[1]

[1] Universitat de Barcelona, Spain
crieramo8@alumnes.ub.edu
camilorey@gmail.com
{epuertas,oriol_pujol}@ub.edu
[2] Bucknell University, United States
thiago.serra@bucknell.edu

Abstract. Neural networks are more expressive when they have multiple layers. In turn, conventional training methods are only successful if the depth does not lead to numerical issues such as exploding or vanishing gradients, which occur less frequently when the layers are sufficiently wide. However, increasing width to attain greater depth entails the use of heavier computational resources and leads to overparameterized models. These subsequent issues have been partially addressed by model compression methods such as quantization and pruning, some of which relying on normalization-based regularization of the loss function to make the effect of most parameters negligible. In this work, we propose instead to use regularization for preventing neurons from dying or becoming linear, a technique which we denote as *jumpstart regularization*. In comparison to conventional training, we obtain neural networks that are thinner, deeper , and—most importantly—more parameter-efficient.

Keywords: Deep learning · Model compression · ReLU networks.

1 Introduction

Leap, and the net will appear.

Anonymous

Artificial neural networks are inspired by the simple, yet powerful idea that predictive models can be produced by combining units that mimic biological neurons. In fact, there is a rich discussion on what should constitute each unit and how the units should interact with one another. Units that work in parallel form a layer, whereas a sequence of layers transforming data unidirectionally define a feedforward network. Deciding the number of such layers—the *depth* of the network—is yet a topic of debate and technical challenges.

A neural network is trained for a particular task by minimizing the loss function associated with a sample of data in order for the network to learn a function of interest. Although several universal approximation results show that

mathematical functions can generally be approximated to arbitrary precision by single-layer feedforward networks, these results rely on using a very large number of units [26,12,43]. Moreover, simple functions such as XOR cannot be exactly represented with a single layer using the most typical units [45].

In fact, it is commonly agreed that depth is important in neural networks [7,38]. In the popular case of feedforward networks in which each unit is a Rectified Linear Unit (ReLU) [21,47,18,38], the neural network models a piecewise linear function [3]. Under the right conditions, the number of such "pieces"—the *linear regions*—may grow exponentially on the depth of the network [48,46,67]. Depending on the total number of units and size of the input, the number of linear regions is maximized with more or less layers [58]. Similarly, there is an active area of study on bounding the number of layers necessary to model any function that a given type of network can represent [45,14,3,20,69,27].

Although shallow networks present competitive accuracy results in some cases [4], deep neural networks have been established as the state-of-the-art over and again in areas such as computer vision and natural language processing [39,29,37,62,24,30,70,13] thanks to the the development and popularization of backpropagation [71,54,41]. However, Stochastic Gradient Descent (SGD) [53]—the training algorithm associated with backpropagation—may have difficulties to converge to a good model due to exploding or vanishing gradients [28,35,49,6].

Exploding and vanishing gradients are often attributed to excessive depth, inadequate choice of parameters for the learning algorithm, or inappropriate scaling between network parameters, inputs, and outputs [17,32]. This issue has also inspired unit augmentations [44,25,60], additional connections across layers [23,30], and output normalization [32,52]. Indeed, it is somewhat intuitive that gradient updates, depth, and parameter scaling may affect one another.

In lieu of reducing depth, we may also increase the number of neurons per layer [72,64,65,66,22]. That leads to models that are considerably more complex, and which are often trained with additional terms in the loss function such as weight normalization to induce simpler models that hopefully generalize better. In turn, that helps model compression techniques such as network pruning methods to remove several parameters with only minor impact to model accuracy.

Nonetheless, vanishing gradients may also be caused by *dead* neurons when using ReLUs. If dead, a ReLU only outputs zero for every sample input. Hence, it does not contribute to updates during training and neither to the expressiveness of the model. To a lesser but relevant extent, similar issues can be observed with a RELU which never outputs zero, which we refer to as a *linear* neuron.

In this work, we aim to reverse neurons which die or become linear during training. Our approach is based on satisfying certain constraints throughout the process. For a margin defined for each unit, at least one input from the sample is above and another input is below. For each layer and input from the sample, at least one unit in the layer has that input above such a margin and another unit has it below. In order to use SGD for training, these constraints are dualized as part of the loss function and thus become a form of regularization that would prevent converging with the original loss function to spurious local minima.

2 Background

We consider a feedforward neural network modeling a function $\hat{\boldsymbol{y}} = f_\theta(\boldsymbol{x})$ with an input layer $\boldsymbol{x} = \boldsymbol{h}^0 = [h_1^0\ h_2^0 \dots h_{n_0}^l]^T$, L hidden layers, and each layer $\ell \in \mathbb{L} = \{1, 2, \dots, L\}$ having n_ℓ units indexed by $i \in \mathbb{N}_\ell = \{1, 2, \dots, n_\ell\}$. For each layer $\ell \in \mathbb{L}$, let $\boldsymbol{W}^\ell$ be the $n_\ell \times n_{\ell-1}$ matrix in which the j-th row corresponds to the weights of neuron j in layer ℓ and $\boldsymbol{b}^\ell$ be vector of biases of layer ℓ. The preactivation output of unit j in layer ℓ is $g_i^\ell = \boldsymbol{W}_j^\ell \boldsymbol{h}^{\ell-1} + b_j^\ell$ and the output is $h_j^\ell = \sigma(g_j^\ell)$ for an activation function σ, which if not nonlinear would allow hidden layer ℓ to be removed by directly connecting layers $\ell - 1$ and $\ell + 1$ [55]. We refer to $\boldsymbol{g}^\ell(\chi)$ and $\boldsymbol{h}^\ell(\chi)$ as the values of $\boldsymbol{g}^\ell$ and $\boldsymbol{h}^\ell$ when $\boldsymbol{x} = \chi$.

For the scope of this work, we consider the ReLU activation function $\sigma(u) = \max\{0, u\}$. Typically, the output of a feedforward neural network is produced by a softmax layer following the last hidden layer [10], $\hat{\boldsymbol{y}} = \rho(\boldsymbol{h}^L)$ with $\rho(\boldsymbol{h}^L)_j = e^{h_j^L} / \sum_{k=1}^{n_L} e^{h_k^L}\ \forall j \in \{1, \dots, n_L\}$, which is a peripheral aspect to our study.

The neural network is trained by minimizing a loss function $\mathcal{L}$ over a parameter set $\theta := \{(\boldsymbol{W}^\ell, \boldsymbol{b}^\ell)\}_{\ell=1}^L$ based on the N samples of a training set $\mathbb{X} := \{(\boldsymbol{x}^i)\}_{i=1}^N$ to yield predictions $\{\hat{\boldsymbol{y}}^i := f_\theta(\boldsymbol{x}^i)\}_{i=1}^N$ that approximate the sample labels $\{\boldsymbol{y}^i\}_{i=1}^N$ using metrics such as least squares or cross entropy [19,59]:

$$\min_\theta \quad \mathcal{L}\left(\theta, \left\{(\hat{\boldsymbol{y}}^i, \boldsymbol{y}^i)\right\}_{i=1}^N\right) \tag{1}$$

$$\text{s.t.} \quad \hat{\boldsymbol{y}^i} = f_\theta(\boldsymbol{x}^i) \qquad \forall i \in \{1, 2, \dots, N\} \tag{2}$$

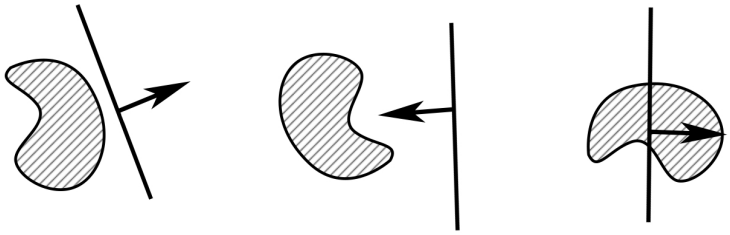
Whereas a neural network is not typically trained through constrained optimization, we believe that our approach is more easily understood under such a mindset, which aligns with further work emerging from this community [8,31,15].

3 Death, Stagnation, and Jumpstarting

According to its pre-activation output, every ReLU is either *inactive* if $g_i^\ell \leq 0$ and thus $h_i^\ell = 0$ or *active* if $g_i^\ell > 0$ and thus $h_i^\ell = g_i^\ell > 0$. If a ReLU does not alternate between those states for different inputs, then the unit is considered *stable* [68] and by consequence the neural network models a less expressive function [56]. In certain cases, those units can be merged or removed without affecting the model [55,57]. We consider in this work a superset of such units—those which do not change of state at least for the training set:

Definition 1. *For a training set $\mathbb{X}$, unit j in layer ℓ is dead if $h_j^\ell(\boldsymbol{x}^i) = 0\ \forall i \in \{1, 2, \dots, N\}$, linear if $h_j^\ell(\boldsymbol{x}^i) > 0\ \forall i \in \{1, 2, \dots, N\}$, or nonlinear otherwise. Layer ℓ dead or linear if all of its units are dead or linear, respectively.*

Figure 1 illustrates geometrically the classification of the unit based on the training set. If dead, a unit impairs the training of the neural network because it always outputs zero for the inputs in the training set. Unless the units preceding



(a) Dead unit (b) Linear unit (c) Nonlinear unit

Fig. 1: A unit j in layer ℓ separates the input space $\boldsymbol{h}^{\ell-1}$ into an open half-space $\boldsymbol{W}_j^\ell \boldsymbol{h}^{\ell-1} + b_j^\ell > 0$ in which the unit is active and a closed half-space $\boldsymbol{W}_j^\ell \boldsymbol{h}^{\ell-1} + b_j^\ell \leq 0$ in which the unit is inactive. The arrow in each case points to the active side. The unit is dead if the inputs from training set $\mathbb{X}$ lie exclusively on the inactive side (1a); linear if exclusively on the active side (1b); and nonlinear otherwise (1c). Note that the partitioning with respect to $\boldsymbol{h}^{\ell-1}$ is linear but it may not necessarily be with respect to the network input $\boldsymbol{x}$ for $\ell > 2$ [58].

a dead unit are updated in such a way that the unit is no longer dead, then the gradients of its output remain at zero and the parameters associated with the dead unit are no longer updated [42,61], which effectively reduces the modeling capacity. If a layer dies, then the training stops because the gradients are zero.

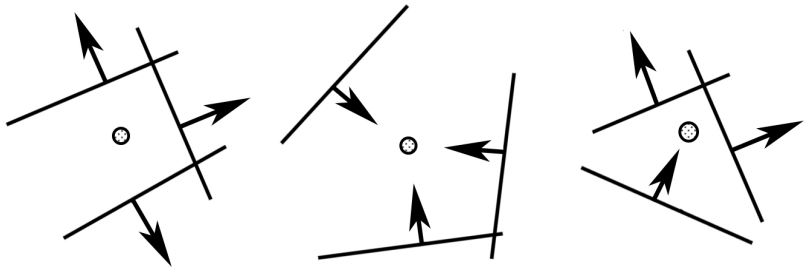
For an intuitive and training-independent discussion, we consider incidence of dead layers at random. If the probability that a unit is dead upon initialization is p, as reasoned in [42], then layer ℓ is dead with probability p^{n_ℓ} and at least one layer is dead with probability $1 - \prod_{\ell=1}^{L}(1-p)^{n_\ell}$. If a layer is too thin or the network is too deep, then the network is more likely to be untrainable. We may discard dead unit initializations, but that ignores the impact on the training set:

Definition 2. *For a hidden layer $\ell \in \mathbb{L}$, an input x is considered a dead point if $\boldsymbol{h}^\ell(x) = 0$, a linear point if $\boldsymbol{h}^\ell(x) > 0$, and a nonlinear point otherwise.*

Figure 2 illustrates geometrically the classification of a point based on which units are activated. If $x^i \in \mathbb{X}$ is a dead point at layer ℓ, then there is no backpropagation associated with x^i to the hidden layers 1 to $\ell - 1$. Hence, its contribution to the training process is diminished unless a subsequent gradient update at a preceding unit reverts this condition. If $\ell = L$, then x^i is effectively not part of the training set. If all points die, regardless of the layer, then training halts.

If we also associate a probability q for $\boldsymbol{x}^i$ not activating a unit, then $\boldsymbol{x}^i$ is dead for layer ℓ with probability q^{n_ℓ} and for at least one layer of the neural network with probability $1 - \prod_{\ell=1}^{L}(1-q)^{n_\ell}$. Unlike p, q is bound to be significant.

We may likewise regard linear units and linear points as less desirable than nonlinear units and nonlinear points. A linear unit limits the expressiveness of the model, since it always contributes the same linear transformation to every



(a) Dead point (b) Linear point (c) Nonlinear point

Fig. 2: Classification of the inputs of a layer ℓ following the same graphical notation as in Figure 1. An input is considered a dead point if it is in the closed half-space $\boldsymbol{W}_j^\ell \boldsymbol{h}^{\ell-1} + b_j^\ell \leq 0$ in which each and every unit $j \in \mathbb{N}_\ell$ is inactive (2a); a linear point if it is in the open half-space $\boldsymbol{W}_j^\ell \boldsymbol{h}^{\ell-1} + b_j^\ell > 0$ in which each and every unit $j \in \mathbb{N}_\ell$ is active (2b); and a nonlinear point otherwise (2c).

input in the training set. A linear point can be more difficult to discriminate from other inputs, in particular if those inputs are also linear points.

Inspired by the prior discussion, we formulate the following constraints:

$$\max_{\boldsymbol{x}^i \in \mathbb{X}} g_j^\ell(\boldsymbol{x}^i) \geq 1 \qquad \forall \ell \in \mathbb{L}, j \in \mathbb{N}_\ell \tag{3}$$

$$\min_{\boldsymbol{x}^i \in \mathbb{X}} g_j^\ell(\boldsymbol{x}^i) \leq -1 \qquad \forall \ell \in \mathbb{L}, j \in \mathbb{N}_\ell \tag{4}$$

$$\max_{j \in \mathbb{N}_\ell} g_j^\ell(\boldsymbol{x}^i) \geq 1 \qquad \forall \ell \in \mathbb{L}, \boldsymbol{x}^i \in \mathbb{X} \tag{5}$$

$$\min_{j \in \mathbb{N}_\ell} g_j^\ell(\boldsymbol{x}^i) \leq -1 \qquad \forall \ell \in \mathbb{L}, \boldsymbol{x}^i \in \mathbb{X} \tag{6}$$

Dead and linear units are respectively prevented by the constraints in (3) and (4). Dead and linear points are prevented by the constraints in (5) and (6).

In order to relax such constraints and rely on SGD, we dualize the constraints in (3)–(6) with a term in the objective function to induce their satisfaction:

$$\min_{\theta} \quad \mathcal{L}\left(\theta, \left\{(\hat{\boldsymbol{y}}^i, \boldsymbol{y}^i)\right\}_{i=1}^{N}\right) + \lambda \mathcal{P}(\xi^+, \xi^-, \psi^+, \psi^-) \tag{7}$$

$$\text{s.t.} \quad \hat{\boldsymbol{y}^i} = f_\theta(\boldsymbol{x}^i) \qquad \forall i \in \{1, 2, \ldots, N\} \tag{8}$$

$$\xi^+_{j\ell} = \max\left\{0, 1 - \max_{\boldsymbol{x}^i \in \mathbb{X}} g^\ell_j(\boldsymbol{x}^i)\right\} \qquad \forall \ell \in \mathbb{L}, j \in \mathbb{N}_\ell \tag{9}$$

$$\xi^-_{j\ell} = \max\left\{0, -1 - \min_{\boldsymbol{x}^i \in \mathbb{X}} g^\ell_j(\boldsymbol{x}^i)\right\} \qquad \forall \ell \in \mathbb{L}, j \in \mathbb{N}_\ell \tag{10}$$

$$\psi^+_{i\ell} = \max\left\{0, 1 - \max_{j \in \mathbb{N}_\ell} g^\ell_j(\boldsymbol{x}^i)\right\} \qquad \forall \ell \in \mathbb{L}, \boldsymbol{x}^i \in \mathbb{X} \tag{11}$$

$$\psi^-_{i\ell} = \max\left\{0, -1 - \min_{j \in \mathbb{N}_\ell} g^\ell_j(\boldsymbol{x}^i)\right\} \qquad \forall \ell \in \mathbb{L}, \boldsymbol{x}^i \in \mathbb{X} \tag{12}$$

We denote by ξ^+, ξ^-, ψ^+, and ψ^- the nonnegative deficits associated with the corresponding constraints in (3)–(6) which are not satisfied. These deficits are combined and weighted against the original loss function $\mathcal{L}$ through a function $\mathcal{P}$, for which we have considered the arithmetic mean as well as the 1 and 2-norms.

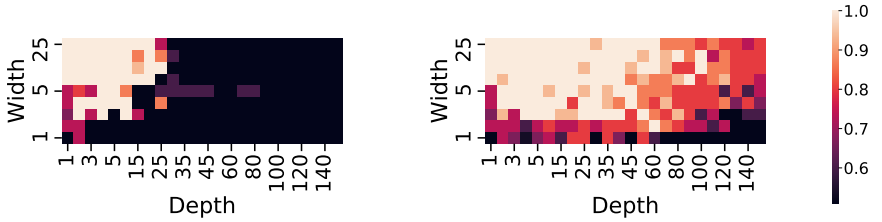
Note that we can approach convolutional neural networks [16,39] with only minor changes to what has been discussed above, since we can regard them as equivalent to a feedforward neural network with parameter sharing and which is not fully connected. The main difference to work with them directly is that the preactivation of the unit becomes a matrix instead of a scalar. We computer the corresponding margin through the maximum or minimum over those values.

4 Computational Experiments

Our first experiment (Figure 3) is based on the `MOONS` dataset [51] with 85 points for training and 15 for validation. We test every width in $\{1, 2, 3, 4, 5, 10, 15, 20, 25\}$ with every depth in $\{1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, \ldots, 150\}$. We chose a simpler dataset to limit the inference of factors such as overfitting, underfitting, or batch size issues. The networks are implemented in `Tensorflow` [1] and `Keras` [11] with Glorot uniform initialization [17] and trained using Adam [34] for 5000 epochs, learning rate of $\epsilon = 0.01$, and batch size of 85. For each depth-width pair, we train a baseline network and a network with jumpstart using 1-norm as the aggregation function $\mathcal{P}$ and loss coefficient $\lambda = 10^{-4}$.

With jumpstart, we successfully train networks of width 3 with a depth up to 60 instead of 10 for the baseline and width 25 with a depth of up to 100 instead of 30. Hence, there is an approximately 5-fold increase in trainable depth.

Our second experiment (Table 1) evaluates convolutional neural networks trained on the MNIST dataset [40]. We test every depth from 2 to 68 in increments of 4 with every width in $\{2, 4, 8\}$, where the width refer to the number of filters per layer. The networks are implemented as before, but with a learning rate of 0.001 over 50 epochs, batch size of 1024, kernel dimensions $(3, 3)$, padding



(a) Validation accuracy with baseline (b) Validation accuracy with jumpstart

Fig. 3: Heatmap contrasting validation accuracy for neural networks trained on `MOONS` with depth between 1 and 150 and width between 1 and 25. The left plot is the baseline and the right plot shows the results when using jumpstart. The accuracy ranges from a low of 0.5 (black) to a high of 1.0 (beige), with the former corresponding to random guessing since the dataset has two balanced classes.

Table 1: Summary of the results for the convolutional neural networks trained on the MNIST dataset without jumpstart (baseline) and with jumpstart.

	Baseline		Jumpstart	
	Training	Validation	Training	Validation
Best overall accuracy	0.999467	0.9885	0.999533	0.9911
Successful model	18	18	54	54
Best for depth-width pair	8	11	45	41

to produce an output of same dimensions as the input, Glorot uniform initialization [17], flattening before the output layer and using a baseline and a jumpstart network with 1-norm as the aggregation function $\mathcal{P}$ and loss coefficient $\lambda = 10^{-8}$.

With jumpstart, we successfully train networks combining all widths and depths in comparison to only up to depth 12 for widths 2 and 4 and only up to depth 24 for width 8 in the baseline. In other words, only 18 baseline network trainings converge, which we denote as the successful models in Table 1.

Our third experiment (Figure 4) evaluates convolutional neural networks trained on the CIFAR-10 dataset [36]. We test every depth in $\{10, 20, 30\}$ with every width in $\{2, 8, 16, 32, 64, 96, 192\}$. The networks are implemented in Pytorch [50], with learning rates $\varepsilon \in \{0.001, 0.0001\}$ over 400 epochs, batch size of 128, kernel dimensions and padding as before, Kaiming uniform initialization [24], global max-avg concat pooling before the output layer, and jumpstart networks with 2-norm (L^2) as one aggregation function $\mathcal{P}$ with loss coefficient $\lambda \in \{0.001, 0.1\}$ as well as the mean ($\bar{x}$) as another $\mathcal{P}$ with $\lambda \in \{0.1, 1\}$.

With jumpstart, we successfully train networks with depth up to 30 in comparison to no more than 20 in the baseline. The best performance—0.766 for jumpstart and 0.734 for baseline—is observed for both with $\varepsilon = 0.001$, where the validation accuracy of each jumpstart experiment exceeds the baseline in 18

to 20 out of 21 depth-width pairs. The baseline is comparatively more competitive with $\varepsilon = 0.0001$, but the overall validation accuracy drops significantly.

5 Conclusion

We have presented a regularization technique for training thinner and deeper neural networks, which leads to a more efficient use of the dataset and to neural networks that are more parameter-efficient. Although massive models are currently widely popular in theory [33] and practice [2], their associated economical barriers and environmental footprint [63] as well as societal impact [5] are known concerns. Hence, we present a potential alternative to lines of work such as model compression [9] by avoiding to operate with larger models. Whereas deeper networks are often pursued, trainable thinner networks are surprisingly not.

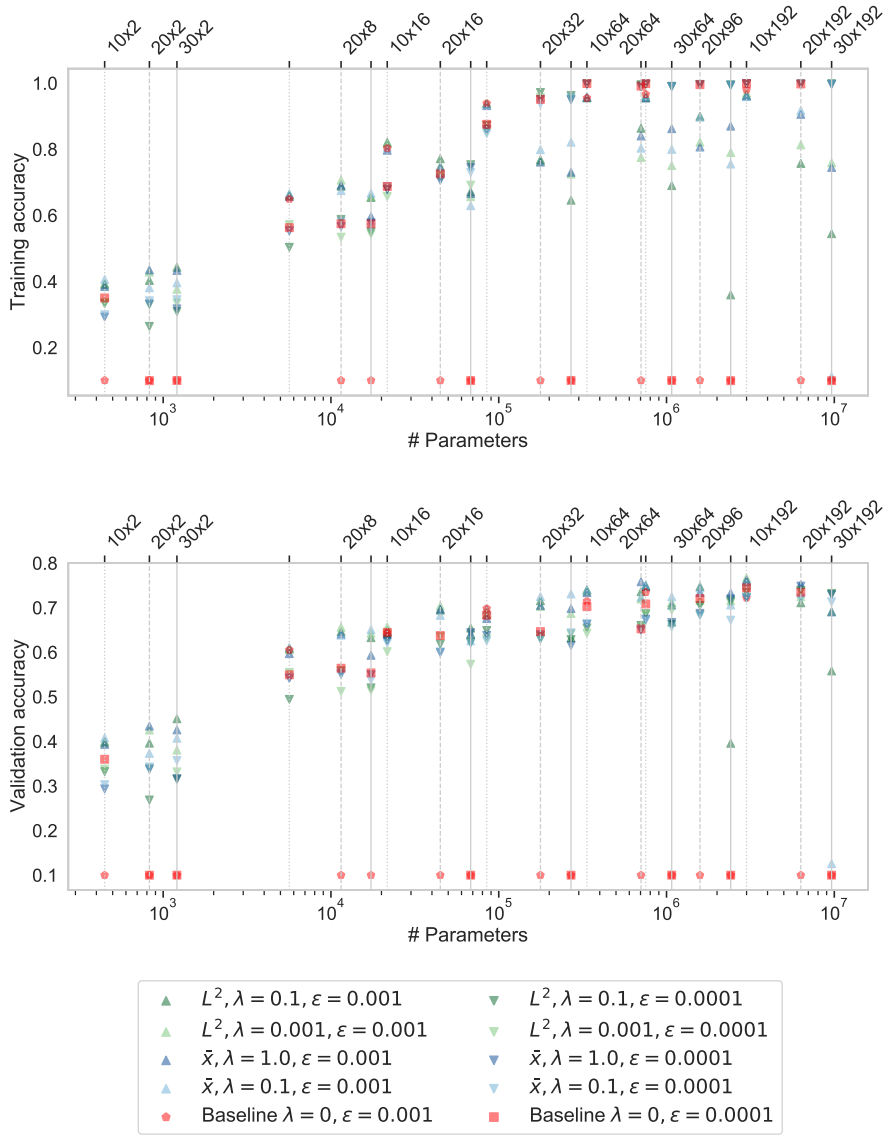


Fig. 4: Scatter chart of number of parameters by accuracy for training (top) and validation (bottom) of convolutional neural networks trained on CIFAR-10. Some depth-width pairs are shown above the plots for reference and the gridlines are solid for depth 30, dashed for 20, and dotted for 10. The results of this experiment are plotted in this format due to their greater variability in comparison to the second experiment, which permits evaluating parameter efficiency. With same number of units but fewer parameters, the results for 20×8 are better than 10×16 and likewise for 20×32 when compared with 10×64.

Acknowledgements

Thiago Serra was supported by the National Science Foundation (NSF) grant IIS 2104583.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), `https://www.tensorflow.org/`, software available from tensorflow.org
2. Amodei, D., Hernandez, D., Sastry, G., Clark, J., Brockman, G., Sutskever, I.: AI and compute. `https://openai.com/blog/ai-and-compute/` (2018), accessed: 2020-12-23
3. Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with rectified linear units. In: ICLR (2018)
4. Ba, J., Caruana, R.: Do deep nets really need to be deep? In: NeurIPS (2014)
5. Bender, E.M., Gebru, T., McMillan-Major, A., Shmitchell, S.: On the dangers of stochastic parrots: Can language models be too big? In: FAccT (2021)
6. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks **5**(2), 157–166 (1994)
7. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives (2014)
8. Bienstock, D., Muñoz, G., Pokutta, S.: Principled deep neural network training through linear programming. CoRR **abs/1810.03218** (2018)
9. Blalock, D., Ortiz, J., Frankle, J., Guttag, J.: What is the state of neural network pruning? In: MLSys (2020)
10. Bridle, J.S.: Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In: Soulié, F.F., Hérault, J. (eds.) Neurocomputing. pp. 227–236. Springer Berlin Heidelberg, Berlin, Heidelberg (1990)
11. Chollet, F., et al.: Keras. `https://keras.io` (2015)
12. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems (MCSS) **2**(4), 303–314 (Dec 1989). https://doi.org/10.1007/BF02551274, `http://dx.doi.org/10.1007/BF02551274`
13. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding (2018), `http://arxiv.org/abs/1810.04805`, cite arxiv:1810.04805Comment: 13 pages
14. Eldan, R., Shamir, O.: The power of depth for feedforward neural networks (2016)
15. Fischetti, M., Stringher, M.: Embedded hyper-parameter tuning by simulated annealing. CoRR **abs/1906.01504** (2019)
16. Fukushima, K., Miyake, S.: Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In: Amari, S.i., Arbib, M.A. (eds.) Competition and Cooperation in Neural Nets. pp. 267–285. Springer Berlin Heidelberg, Berlin, Heidelberg (1982)

17. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics (2010)
18. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: AISTATS (2011)
19. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
20. Gribonval, R., Kutyniok, G., Nielsen, M., Voigtlaender, F.: Approximation spaces of deep neural networks (2020)
21. Hahnloser, R., Sarpeshkar, R., Mahowald, M., Douglas, R., Seung, S.: Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature **405** (2000)
22. Hasanpour, S.H., Rouhani, M., Fayyaz, M., Sabokrou, M., Adeli, E.: Towards principled design of deep convolutional networks: Introducing simpnet. CoRR **abs/1802.06205** (2018), http://arxiv.org/abs/1802.06205
23. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015), http://arxiv.org/abs/1512.03385
24. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015 IEEE International Conference on Computer Vision (ICCV) pp. 1026–1034 (2015)
25. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. CoRR **abs/1502.01852** (2015), http://arxiv.org/abs/1502.01852
26. Hecht-Nielsen, R.: Kolmogorov's mapping neural network existence theorem. In: Proceedings of the international conference on Neural Networks. vol. 3, pp. 11–14. IEEE Press New York (1987)
27. Hertrich, C., Basu, A., Summa, M.D., Skutella, M.: Towards lower bounds on the depth of relu neural networks (2021)
28. Hochreiter, S.: Untersuchungen zu dynamischen neuronalen netzen. Diploma, Technische Universität München **91**(1) (1991)
29. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
30. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: CVPR. pp. 2261–2269. IEEE Computer Society (2017), http://dblp.uni-trier.de/db/conf/cvpr/cvpr2017.html#HuangLMW17
31. Icarte, R., Illanes, L., Castro, M., Cire, A., McIlraith, S., Beck, C.: Training binarized neural networks using MIP and CP. In: International Conference on Principles and Practice of Constraint Programming (CP) (2019)
32. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR **abs/1502.03167** (2015), http://arxiv.org/abs/1502.03167
33. Jacot, A., Gabriel, F., Hongler, C.: Neural tangent kernel: Convergence and generalization in neural networks. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. p. 8580–8589. NIPS'18, Curran Associates Inc., Red Hook, NY, USA (2018)
34. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014), http://arxiv.org/abs/1412.6980
35. Kolen, J.F., Kremer, S.C.: Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies, pp. 237–243. Wiley-IEEE Press (2001). https://doi.org/10.1109/9780470544037.ch14

36. Krizhevsky, A.: Learning multiple layers of features from tiny images pp. 32–33 (2009), https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
37. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012), http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
38. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (May 2015). https://doi.org/10.1038/nature14539, http://dx.doi.org/10.1038/nature14539
39. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE. vol. 86, pp. 2278–2324 (1998), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665
40. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010), http://yann.lecun.com/exdb/mnist/
41. LeCun, Y., Touresky, D., Hinton, G., Sejnowski, T.: A theoretical framework for back-propagation. In: Proceedings of the 1988 connectionist models summer school. vol. 1, pp. 21–28 (1988)
42. Lu, L., Shin, Y., Su, Y., Karniadakis, G.E.: Dying relu and initialization: Theory and numerical examples. arXiv preprint arXiv:1903.06733 (2019)
43. Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L.: The expressive power of neural networks: A view from the width (2017)
44. Maas, A.L., Hannun, A.Y., Ng, A.Y.: Rectifier nonlinearities improve neural network acoustic models. In: in ICML Workshop on Deep Learning for Audio, Speech and Language Processing (2013)
45. Minsky, M., Papert, S.: Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge, MA, USA (1969)
46. Montúfar, G., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. In: NeurIPS (2014)
47. Nair, V., Hinton, G.: Rectified linear units improve restricted boltzmann machines. In: ICML (2010)
48. Pascanu, R., Montúfar, G., Bengio, Y.: On the number of response regions of deep feedforward networks with piecewise linear activations. In: ICLR (2014)
49. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks (2013)
50. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
51. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

52. Pooladian, A., Finlay, C., Oberman, A.M.: Farkas layers: don't shift the data, fix the geometry. CoRR **abs/1910.02840** (2019), http://arxiv.org/abs/1910.02840
53. Robbins, H., Monro, S.: A stochastic approximation method. The Annals of Mathematical Statistics **22**(3), 400–407 (1951)
54. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**, 533–536 (1986)
55. Serra, T., Kumar, A., Ramalingam, S.: Lossless compression of deep neural networks. In: CPAIOR (2020)
56. Serra, T., Ramalingam, S.: Empirical bounds on linear regions of deep rectifier networks. In: AAAI (2020)
57. Serra, T., Kumar, A., Yu, X., Ramalingam, S.: Scaling up exact neural network compression by relu stability (2021)
58. Serra, T., Tjandraatmadja, C., Ramalingam, S.: Bounding and counting linear regions of deep neural networks (2018)
59. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, USA (2014)
60. Shang, W., Sohn, K., Almeida, D., Lee, H.: Understanding and improving convolutional neural networks via concatenated rectified linear units. CoRR **abs/1603.05201** (2016), http://arxiv.org/abs/1603.05201
61. Shin, Y., Karniadakis, G.E.: Trainability and data-dependent initialization of over-parameterized relu neural networks. CoRR **abs/1907.09696** (2019), http://arxiv.org/abs/1907.09696
62. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014), http://arxiv.org/abs/1409.1556
63. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: ACL (2019)
64. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S.E., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. CoRR **abs/1409.4842** (2014), http://arxiv.org/abs/1409.4842
65. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. CoRR **abs/1905.11946** (2019), http://arxiv.org/abs/1905.11946
66. Tan, M., Le, Q.V.: Efficientnetv2: Smaller models and faster training. CoRR **abs/2104.00298** (2021), https://arxiv.org/abs/2104.00298
67. Telgarsky, M.: Representation benefits of deep feedforward networks. CoRR **abs/1509.08101** (2015)
68. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: ICLR (2019)
69. Vardi, G., Reichman, D., Pitassi, T., Shamir, O.: Size and depth separation in approximating benign functions with neural networks (2021)
70. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
71. Werbos, P.J.: Applications of advances in nonlinear sensitivity analysis. In: Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC. pp. 762–770 (1981)
72. Zagoruyko, S., Komodakis, N.: Wide residual networks. CoRR **abs/1605.07146** (2016), http://arxiv.org/abs/1605.07146