Backprop Evolution

Maximilian Alber [*] [†] MAXIMILIAN.ALBER@TU-BERLIN.DE
TU Berlin

Irwan Bello [*] IBELLO@GOOGLE.COM
Barret Zoph BARRETZOPH@GOOGLE.COM
Pieter-Jan Kindermans [‡] PIKINDER@GOOGLE.COM
Prajit Ramachandran [‡] PRAJIT@GOOGLE.COM
Quoc Le QVL@GOOGLE.COM
Google Brain

arXiv:1808.02822v1 [cs.LG] 8 Aug 2018

Abstract

The back-propagation algorithm is the cornerstone of deep learning. Despite its importance, few variations of the algorithm have been attempted. This work presents an approach to discover new variations of the back-propagation equation. We use a domain specific language to describe update equations as a list of primitive functions. An evolution-based method is used to discover new propagation rules that maximize the generalization performance after a few epochs of training. We find several update equations that can train faster with short training times than standard back-propagation, and perform similar as standard back-propagation at convergence.

Keywords: Back-propagation, neural networks, automl, meta-learning.

1. Introduction

The back-propagation algorithm is one of the most important algorithms in machine learning (Linnainmaa (1970); Werbos (1974); Rumelhart et al. (1986)). A few attempts have been made to change the back-propagation equation with some degrees of success (e.g., Bengio et al. (1994); Lillicrap et al. (2014); Lee et al. (2015); Nøkland (2016); Liao et al. (2016)). Despite these attempts, modifications of back-propagation equations have not been widely used as these algorithms rarely improve practical applications, and sometimes hurt them.

Inspired by the recent successes of automated search methods for machine learning (Zoph and Le, 2017; Zoph et al., 2018; Bello et al., 2017; Brock et al., 2017; Real et al., 2018; Bender et al., 2018a), we propose a method for automatically generating back-propagation equations. To that end, we introduce a domain specific language to describe such mathematical formulas as a list of primitive functions and use an evolution-based method to discover new propagation rules. The search is conditioned to maximize the generalization after a few epochs of training. We find several variations of the equation that can work as well as the standard back-propagation equation. Furthermore, several variations can achieve improved accuracies with short training times. This can be used to improve algorithms

[*] Contributed equally.
[†] Work was done as intern at Google Brain.
[‡] Work was done as a member of the Google AI Residency program (g.co/airesidency).

© 2018 M. Alber, I. Bello, B. Zoph, P.-J. Kindermans, P. Ramachandran & Q. Le.

like Hyperband (Li et al., 2017) which make accuracy-based decisions over the course of training.

2. Backward propagation

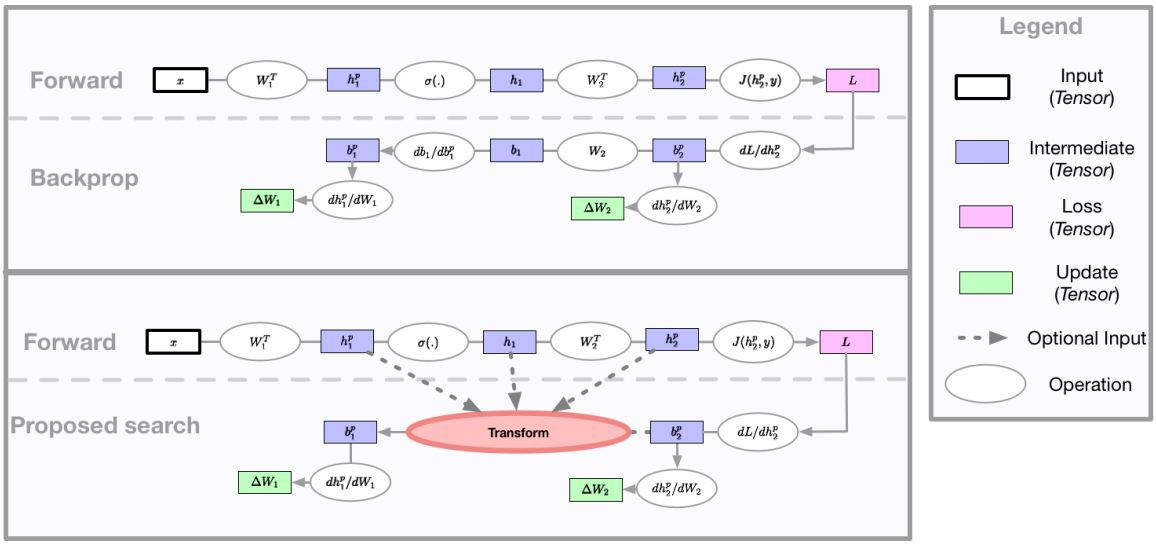

Figure 1: Neural networks can be seen as computational graphs. The forward graph is defined by the network designer, while the back-propagation algorithm implicitly defines a computational graph for the parameter updates. Our main contribution is exploring the use of evolution to find a computational graph for the parameter updates that is more effective than standard back-propagation.

The simplest neural network can be defined as a sequence of matrix multiplications and nonlinearities:

$$h_i^p = W_i^T h_{i-1}, \qquad h_i = \sigma(h_i^p).$$

where $h_0 = x$ is the input to the network, i indexes the layer and W_i is the weight matrix of the i-th layer. To optimize the neural network, we compute the partial derivatives of the loss $J(f(x), y)$ w.r.t. the weight matrices $\frac{\partial J(f(x),y)}{\partial W_i}$. This quantity can be computed by making use of the chain rule in the back-propagation algorithm. To compute the partial derivative with respect to the hidden activations $b_i^p = \frac{\partial J(f(x),y)}{h_i^p}$, a sequence of operations is applied to the derivatives:

$$b_L = \frac{\partial J(f(x), y)}{\partial h_L}, \qquad b_{i+1}^p = b_{i+1} \frac{\partial h_{i+1}}{\partial h_{i+1}^p}, \qquad b_i = b_{i+1}^p \frac{\partial h_{i+1}^p}{\partial h_i}. \tag{1}$$

Once b_i^p is computed, the weights update can be computed as: $\Delta W_i = b_i^p \frac{\partial h_i^p}{W_i}$.

As shown in Figure 1, the neural network can be represented as a forward and backward computational graph. Given a forward computational graph defined by a network designer, the back-propagation algorithm defines a backward computational graph that is

used to update the parameters. However, it may be possible to find an improved backward computational graph that results in better *generalization*.

Recently, automated search methods for machine learning have achieved strong results on a variety of tasks (e.g., Zoph and Le (2017); Baker et al. (2017); Real et al. (2017); Bello et al. (2017); Brock et al. (2017); Ramachandran et al. (2018); Zoph et al. (2018); Bender et al. (2018b); Real et al. (2018)). With the exception of Bello et al. (2017), these methods involve modifying the forward computational graph, relying on back-propagation to define the appropriate backward graph. In this work, we instead concern ourselves with modifying the backward computational graph and use a search method to find better formulas for b_i^p, yielding new training rules.

3. Method

In order to discover improved update rules, we employ an evolution algorithm to search over the space of possible update equations. At each iteration, an evolution controller sends a batch of mutated update equations to a pool of workers for evaluation. Each worker trains a fixed neural network architecture using its received mutated equation and reports the obtained validation accuracy back to our controller.

3.1. Search Space

We use a domain-specific language (DSL) inspired by Bello et al. (2017) to describe the equations used to compute b_i^p. The DSL expresses each b_i^p equation as $f(u_1(op_1), u_2(op_2))$ where op_1, op_2 are possible operands, $u_1(\cdot)$ and $u_2(\cdot)$ are unary functions, and $f(\cdot,\cdot)$ is a binary function. The sets of unary functions and binary functions are manually specified but individual choices of functions and operands are selected by the controller. Examples of each component are as follows:

- **Operands:** W_i (weight matrix of the current layer), R_i (Gaussian matrix), R_{Li} (Gaussian random matrix mapping from b_L^p to b_i^p), h_i^p, h_i, h_{i+1}^p (hidden activations of the forward propagation), b_L^p, b_{i+1}^p (backward propagated values).
- **Unary functions** $u(x)$**:** x (identity), x^t (transpose), $1/x$, x^2, $\text{sgn}(x)x^2$, x^3, ax, $x+b$, $\text{drop}_d(x)$ (dropout with drop probability $d \in (0.01, 0.1, 0.3)$), $\text{clip}_c(x)$ (clip values in range $[-c, c]$ and $c \in (0.01, 0.1, 0.5, 1.0)$), $x/\|.\|_{fro}$, $x/\|.\|_1$, $x/\|.\|_{-inf}$, $x/\|.\|_{inf}$ (normalizing term by matrix norm).
- **Binary functions** $f(x, y)$**:** $x+y$, $x-y$, $x \odot y$, x/y (element-wise addition, subtraction, multiplication, division), $x \cdot y$ (matrix multiplication) and x (keep left), $\min(x, y)$, $\max(x, y)$ (minimum and maximum of x and y).

where i indexes the current layer. The full set of components used in our experiments is specified in Appendix A. The resulting quantity $f(u_1(op_1), u_2(op_2))$ is then either used as b_i^p in Equation 1 or used recursively as op_1 in subsequent parts of the equation. In our experiments, we explore equations composed of between 1 and 3 binary operations. This DSL is simple but can represent complex equations such as normal back-propagation, feedback alignment (Lillicrap et al., 2014), and direct feedback alignment (Nøkland, 2016).

3.2. Evolution algorithm

The evolutionary controller maintains a population of discovered equations. At each iteration, the controller does one of the following: 1) With probability p, the controller chooses an equation randomly at uniform within the N most competitive equations found so far during the search, 2) With probability $1 - p$, it chooses an equation randomly at uniform from the rest of the population. The controller subsequently applies k mutations to the selected equation, where k is drawn from a categorical distribution. Each of these k mutations simply consists in selecting one of the equation components (e.g., an operand, an unary, or a binary function) uniformly at random and swapping it with another component of the same category chosen uniformly at random. Certain mutations lead to mathematically infeasible equations (e.g., a shape mismatch when applying a binary operation to two matrices). When this is the case, the controller restarts the mutation process until successful. N, p and the categorical distribution from which k is drawn are hyperparameters of the algorithm.

To create an initial population, we simply sample N equations at random from the search space. Additionally, in some of our experiments, we instead start with a small population of pre-defined equations (typically the normal back-propagation equation or its feedback alignment variants). The ability to start from existing equations is an advantage of evolution over reinforcement learning based approaches (Zoph and Le (2017); Zoph et al. (2018); Bello et al. (2017); Ramachandran et al. (2018)).

4. Experiments

The choice of model used to evaluate each proposed update equation is an important setting in our method. Larger, deeper networks are more realistic but take longer to train, whereas small models are faster to train but may lead to discovering update equations that do not generalize. We strike a balance between the two criteria by using Wide ResNets (WRN) (Zagoruyko and Komodakis, 2016) with 16 layers and a width multiplier of 2 (WRN 16-2) to train on the CIFAR-10 dataset.

We experiment with different search setups, such as varying the subsets of the available operands and operations and using different optimizers. The evolution controller searches for the update equations that obtain the best accuracy on the validation set. We then collect the 100 best discovered equations, which were determined by the mean accuracy of 5 reruns. Finally, the best equations are used to train models on the full training set, and we report the test accuracy. The experiment setup is further detailed in Appendix B.

4.1. Baseline search and generalization

In the first search we conduct, the controller proposes update equations to train WRN 16-2 networks for 20 epochs with SGD with or without momentum. The top 100 update equations according to validation accuracy are collected and then tested on different scenarios:

(A1) WRN 16-2 for 20 epochs, replicating the search settings.

(A2) WRN 28-10 for 20 epochs, testing generalization to larger models (WRN 28-10 has 10 times more parameters than WRN 16-2).

SGD		SGD and Momentum	
(A1) Search validation			
baseline g_i^p	77.11 ± 3.53	*baseline* g_i^p	83.00 ± 0.63
$\min(g_i^p/\|\|\cdot\|\|_{fro}), \text{clip}_{1.0}(h_i))$	$\mathbf{84.48 \pm 0.45}$	$(g_i^p/\|\|\cdot\|\|_2^{elem})/(2 + \hat{\text{m}}((\text{b}_\text{L}^\text{p} \cdot \text{R}_\text{Li}) \odot \frac{\partial \text{h}_\text{i}}{\partial \text{h}_\text{i}^\text{p}}))$	$\mathbf{85.43 \pm 1.59}$
$(g_i^p/\|\|\cdot\|\|_2^{row}) + \hat{\text{m}}(\frac{\partial \text{h}_\text{i}}{\partial \text{h}_\text{i}^\text{p}})/\|\|\cdot\|\|_0^{elem}$	$\mathbf{84.41 \pm 1.37}$	$(g_i^p/\|\|\cdot\|\|_{fro}) + 0.5 g_i^p$	$\mathbf{85.36 \pm 1.41}$
$g_i^p/\|\|\cdot\|\|_{fro}$	$\mathbf{84.15 \pm 0.66}$	$g_i^p/\|\|\cdot\|\|_{fro}$	$\mathbf{84.23 \pm 0.88}$
$g_i^p/\|\|\cdot\|\|_2^{elem}$	$\mathbf{83.16 \pm 0.90}$	$g_i^p/\|\|\cdot\|\|_2^{elem}$	$\mathbf{83.83 \pm 1.27}$
(A2) Generalize to WRN 28x10			
baseline g_i^p	73.10 ± 1.41	*baseline* g_i^p	79.53 ± 2.89
$(g_i^p/\|\|\cdot\|\|_{inf}^{elem}) \odot (\hat{\text{s}}(\frac{\partial \text{h}_\text{i}}{\partial \text{h}_\text{i}^\text{p}})/\|\|\cdot\|\|_1)$	$\mathbf{88.22 \pm 0.55}$	$\max((\text{b}_{\text{i+1}}^\text{p} \frac{\partial \text{h}_{\text{i+1}}^\text{p}}{\partial \text{h}_\text{i}} - 10.0), g_i^p/\|\|\cdot\|\|_2^{elem})$	$\mathbf{89.43 \pm 0.99}$
$\text{clip}_{0.01}(0.01 + h_i^p - (h_i^p)^+) \odot (g_i^p/\|\|\cdot\|\|_{inf}^{elem})$	$\mathbf{87.28 \pm 0.29}$	$(\text{b}_{\text{i+1}}^\text{p} \frac{\partial \text{h}_{\text{i+1}}^\text{p}}{\partial \text{h}_\text{i}} - 0.01) + (g_i^p/\|\|\cdot\|\|_2^{elem})$	$\mathbf{89.26 \pm 0.67}$
$g_i^p/\|\|\cdot\|\|_{fro}$	$\mathbf{87.17 \pm 0.87}$	$g_i^p/\|\|\cdot\|\|_{fro}$	$\mathbf{89.63 \pm 0.32}$
$g_i^p/\|\|\cdot\|\|_2^{elem}$	$\mathbf{85.30 \pm 1.04}$	$g_i^p/\|\|\cdot\|\|_2^{elem}$	$\mathbf{89.05 \pm 0.88}$
(A3) Generalize to longer training			
baseline g_i^p	92.38 ± 0.10	*baseline* g_i^p	93.75 ± 0.15
$(g_i^p/\|\|\cdot\|\|_2^{elem}) \odot \text{sgn}(\text{bn}(\frac{\partial \text{h}_\text{i}}{\partial \text{h}_\text{i}^\text{p}}))$	$\mathbf{92.97 \pm 0.18}$	$\text{drop}_{0.01}(g_i^p) - (\text{bn}(\text{b}_\text{L}^\text{p} \cdot \text{R}_\text{Li})/\|\|\cdot\|\|_0^{elem})$	93.72 ± 0.20
$(g_i^p/\|\|\cdot\|\|_2^{elem}) - (\hat{\text{m}}(\text{b}_{\text{i+1}}^\text{p} \frac{\partial \text{h}_{\text{i+1}}^\text{p}}{\partial \text{h}_\text{i}})/\|\|\cdot\|\|_{inf})$	$\mathbf{92.90 \pm 0.13}$	$(1 + \text{gnoise}_{0.01}) g_i^p + (g_i^p p1_b/\|\|\cdot\|\|_1^{elem})$	93.66 ± 0.12
$g_i^p/\|\|\cdot\|\|_{fro}$	$\mathbf{92.85 \pm 0.14}$	$g_i^p/\|\|\cdot\|\|_{fro}$	93.41 ± 0.18
$g_i^p/\|\|\cdot\|\|_2^{elem}$	$\mathbf{92.78 \pm 0.13}$	$g_i^p/\|\|\cdot\|\|_2^{elem}$	93.35 ± 0.15
(B1) Searching with longer training			
baseline g_i^p	87.13 ± 0.25	*baseline* g_i^p	88.94 ± 0.11
$(g_i^p/\|\|\cdot\|\|_1) + (\text{b}_{\text{i+1}}^\text{p} \frac{\partial \text{h}_{\text{i+1}}^\text{p}}{\partial \text{h}_\text{i}} /\|\|\cdot\|\|_2^{elem})$	$\mathbf{87.94 \pm 0.22}$	$2 g_i^p + (\text{bn}(\text{b}_{\text{i+1}}^\text{p} \frac{\partial \text{h}_{\text{i+1}}^\text{p}}{\partial \text{h}_\text{i}})/\|\|\cdot\|\|_1^{elem})$	89.04 ± 0.25
$(g_i^p/\|\|\cdot\|\|_1) \odot \text{clip}_{0.1}(\frac{\partial \text{h}_\text{i}}{\partial \text{h}_\text{i}^\text{p}})$	$\mathbf{87.88 \pm 0.39}$	$(\frac{\partial \text{h}_\text{i}}{\partial \text{h}_\text{i}^\text{p}} - 0.5) \odot 2.0 g_i^p$	88.95 ± 0.16
$(g_i^p/\|\|\cdot\|\|_2^{elem}) \odot \text{sgn}(\text{bn}(\frac{\partial \text{h}_\text{i}}{\partial \text{h}_\text{i}^\text{p}}))$	$\mathbf{87.82 \pm 0.19}$	$(\hat{\text{s}}(\text{b}_{\text{i+1}}^\text{p} \frac{\partial \text{h}_{\text{i+1}}^\text{p}}{\partial \text{h}_\text{i}})/\|\|\cdot\|\|_{inf}^{elem}) * 2.0 g_i^p$	88.94 ± 0.20
$(0.5 g_i^p)/(\hat{\text{s}}(\frac{\partial \text{h}_\text{i}}{\partial \text{h}_\text{i}^\text{p}}) + 0.1)$	$\mathbf{87.72 \pm 0.25}$	$(\frac{\partial \text{h}_\text{i}}{\partial \text{h}_\text{i}^\text{p}})^+ \odot \text{clip}_{1.0}(\text{b}_{\text{i+1}}^\text{p} \frac{\partial \text{h}_{\text{i+1}}^\text{p}}{\partial \text{h}_\text{i}})$	88.93 ± 0.14

Table 1: Results of the experiments. For A1-3 we show the two best performing equations on each setup and two equations that consistently perform well across all setups. For B1 we show the four best performing equations. All results are the average test accuracy over 5 repetitions. Baseline is gradient back-propagation. Numbers that are at least 0.1% better are in bold. A description of the operands and operations can be found in Appendix A. We denote $\text{b}_{\text{i+1}}^\text{p} \frac{\partial \text{h}_{\text{i+1}}^\text{p}}{\partial \text{h}_\text{i}^\text{p}}$ with g_i^p.

(A3) WRN 16-2 for 100 epochs, testing generalization to longer training regimes.

The results are listed in Table 1. When evaluated on the search setting (A1), it is clear that the search finds update equations that outperform back-propagation for both SGD and SGD with momentum, demonstrating that the evolutionary controller can successfully find update equations that achieve better accuracies. The update equations also successfully generalize to the larger WRN 28-10 model (A2), outperforming the baseline by up to 15% for SGD and 10% for SGD with momentum. This result suggests that the usage of smaller

models during searches is appropriate because the update equations still generalize to larger models.

However, when the model is trained for longer (A3), standard back-propagation and the discovered update equations perform similarly. The discovered update equations can then be seen as equations that speed up training initially but do not affect final model performance, which can be practical in settings where decisions are made during the early stages of training to continue the experiment, such as some hyperparameter search schemes (Li et al., 2017).

4.2. Searching for longer training times

The previous search experiment finds update equations that work well at the beginning of training but do not outperform back-propagation at convergence. The latter result is potentially due to the mismatch between the search and the testing regimes, since the search used 20 epochs to train child models whereas the test regime uses 100 epochs.

A natural followup is to match the two regimes. In the second search experiment, we train each child model for 100 epochs. To compensate for the increase in experiment time due to training for more epochs, a smaller network (WRN 10-1) is used as the child model. The use of smaller models is acceptable since update equations tend to generalize to larger, more realistic models (see (A2)).

The results are shown in Table 1 section (B1) and are similar to (A3), i.e., we are able to find update rules that perform moderately better for SGD, but the results for SGD with momentum are comparable to the baseline. The similarity between results of (A3) and (B1) suggest that the training time discrepancy may not be the main source of error. Furthermore, SGD with momentum is fairly unchanging to different update equations. Future work can analyze why adding momentum increases robustness.

5. Conclusion and future work

In this work, we presented a method to automatically find equations that can replace standard back-propagation. We use an evolutionary controller that operates in a space of equation components and tries to maximize the generalization of trained networks. The results of our exploratory study show that for specific scenarios, there are equations that yield better generalization performance than this baseline, but more work is required to find an equation that performs better in general scenarios. It is left to future work to distill patterns from equations found by the search, and research under which conditions and why they yield better performance.

Acknowledgements: We would like to thank Gabriel Bender for his technical advice throughout this work and Simon Kornblith for his valuable feedback on the manuscript.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, 2017.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 549–558, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018a. PMLR. URL http://proceedings.mlr.press/v80/bender18a.html.

Gabriel Bender, Pieter-jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Demystifying one-shot architecture search. In *International Conference on Machine Learning*, 2018b.

Samy Bengio, Yoshua Bengio, and Jocelyn Cloutier. Use of genetic programming for the search of a new learning rule for neural networks. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 324–327. IEEE, 1994.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. SMASH: one-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2017.

François Chollet et al. Keras, 2015.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *International Conference on Learning Representations*, 2017.

Qianli Liao, Joel Z Leibo, and Tomaso A Poggio. How important is weight symmetry in backpropagation? In *AAAI*, 2016.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, 1970.

Ilya Loshchilov and Frank Hutter. Sgdr: stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 1037–1045, 2016.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. In *International Conference on Learning Representations (Workshop)*, 2018.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

Paul Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, 1974.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Appendix A. Search space

The operands and operations to populate our search space are the following (i indexes the current layer):

- **Operands:**
  - W_i, $\text{sgn}(W_i)$ (weight matrix of the current layer and sign of it),
  - R_i , S_i (Gaussian and Bernoulli random matrices, same shape as W_i),
  - R_{Li} (Gaussian random matrix mapping from b_L^p to b_i^p),
  - h_i^p, h_i, h_{i+1}^p (hidden activations of the forward propagation),
  - b_L^p, b_{i+1}^p (backward propagated values),
  - $b_{i+1}^p \frac{\partial h_{i+1}^p}{\partial h_i}$, $b_{i+1}^p \frac{\partial h_{i+1}^p}{\partial h_i^p}$ (backward propagated values according to gradient backward propagation),
  - $b_{i+1}^p \cdot R_i$, $(b_{i+1}^p \cdot R_i) \odot \frac{\partial h_i}{\partial h_i^p}$ (backward propagated values according to feedback alignment),
  - $b_L^p \cdot R_{Li}$, $(b_L^p \cdot R_{Li}) \odot \frac{\partial h_i}{\partial h_i^p}$ (backward propagated values according to direct feedback alignment).
- **Unary functions $u(x)$:**
  - x (identity), x^t (transpose), $1/x$, $|x|$, $-x$ (negation), $\mathbb{1}_{x>0}$ (1 if x greater than 0), x^+ (ReLU), $\text{sgn}(x)$ (sign), $\sqrt{|x|}$, $\text{sgn}(x)\sqrt{|x|}$, x^2, $\text{sgn}(x)x^2$, x^3, ax, $x+b$,
  - $x + \text{gnoise}(g)$, $x \odot (1 + \text{gnoise}(g))$ (add or multiply with Gaussian noise of scale $g \in (0.01, 0.1, 0.5, 1.0)$),
  - $\text{drop}_d(x)$ (dropout with drop probability $d \in (0.01, 0.1, 0.3)$), $\text{clip}_c(x)$ (clip values in range $[-c, c]$ and $c \in (0.01, 0.1, 0.5, 1.0)$,
  - $x/\|.\|_0^{elem}$, $x/\|.\|_1^{elem}$, $x/\|.\|_2^{elem}$, $x/\|.\|_{-inf}^{elem}$, $x/\|.\|_{inf}^{elem}$ (normalizing term by vector norm on flattened matrix),
  - $x/\|.\|_0^{col}$, $x/\|.\|_1^{col}$, $x/\|.\|_2^{col}$, $x/\|.\|_{-inf}^{col}$, $x/\|.\|_{inf}^{col}$, $x/\|.\|_0^{row}$, $x/\|.\|_1^{row}$, $x/\|.\|_2^{row}$, $x/\|.\|_{-inf}^{row}$, $x/\|.\|_{inf}^{row}$,(normalizing term by vector norm along columns or rows of matrix),
  - $x/\|.\|_{fro}$, $x/\|.\|_1$, $x/\|.\|_{-inf}$, $x/\|.\|_{inf}$ (normalizing term by matrix norm),
  - $(x - \hat{m}(x))/\sqrt{\hat{s}(x^2)}$ (normalizing with running averages with factor $r = 0.9$).
- **Binary functions $f(x, y)$:**
  - $x + y$, $x - y$, $x \odot y$, x/y (element-wise addition, subtraction, multiplication, division),
  - $x \cdot y$ (matrix multiplication),
  - x (keep left),
  - $\min(x, y)$, $\max(x, y)$ (minimum and maximum of x and y).

Additionally, we add for each operand running averages for the mean and standard deviation as well as a normalize version of it, i.e., subtract the mean and divide by the standard deviation.

So far we described our setup for dense layers. Many state-of-the-art neural networks are additionally powered by convolutional layers, therefore we chose to use convolutional neural networks. Conceptually, dense and convolutional layers are very similar, e.g., convolutional layers can be mimicked by dense layers by extracting the the relevant image patches and performing a matrix dot product. For performance reasons we do to use this technique, but rather map the matrix multiplication operations to corresponding convolutional operations. In this case we keep the (native) 4-dimensional tensors used in convoluational layers and, when required, reshape them to matrices by joining all axes but the sample axis, i.e., join width, height, and filter axes.

Appendix B. Experimental details

The Wide ResNets used in the experiments are not just composed of dense and convolutional layers, but have operations like average pooling. For parts of the backward computational graph that are not covered by the search space, we use the standard gradient equations to propagate backwards.

Throughout the paper we use the CIFAR-10 dataset and use the preprocessing described in Loshchilov and Hutter (2016). The hyperparameter setup is also based on that in Loshchilov and Hutter (2016), and we only modify the learning rate and the optimizer. For experiments with more than 50 epochs, we use cosine decay with warm starts, where warm up phase lasts for 10% of the training steps and the cosine decay reduces the learning rate over the remaining time.

The evolution process is parameterized as follows. With probability $p = 0.7$ we choose an equation out of the $N = 1000$ most competitive equations. For all searches we always modify one operand or one operation. We use Keras (Chollet et al., 2015) and Tensorflow (Abadi et al., 2016) to implement our setup. We typically use 500 workers which run on CPUs for search experiments, which lead to the search converging within 2-3 days.

We experiment with different search setups, i.e., found it useful to use a different subset of the available operands and operations, to select different learning rates as well as optimizers. We use either WRNs with depth 16 and width 2 or depth 10 and width 1 and use four different learning rates. Additionally, during searches we use early stopping on the validation set to reduce the overhead of bad performing equations. Either SGD or SGD with momentum (with the momentum coefficient set to 0.9) is used as the optimizer. The controller tries to maximize the accuracy on the validation set. The top 100 equations are trained on the entire training set, and the test set accuracy is reported in Table 1.