Deep Neuroevolution with Shared Parameters: Ensembling a Better Solution

Michael Painter
Department of Computer Science, Stanford University

Introduction

The normal workflow of using neural networks for a machine learning tasks consists of trying and testing lots of architectures. However, training each from scratch can be slow and time-consuming.

Chen *et al.* [1] utilize knowledge transfer by developing function preserving transforms to speed up this process.

Our work first develops new function preserving transforms, which are used to dynamically widen/deepen a neural network, without altering the function represented by the network.

The process of training a network to convergence can be sped up by utilizing our transforms (at the cost of sample complexity).

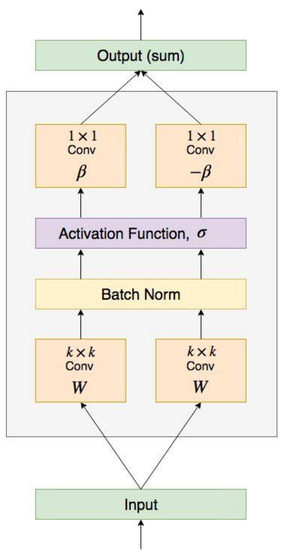
Finally, the transforms do not alter existing weights in the neural network, which is used to implement an efficient architecture search.

Zero-Initialization

Our function preserving transforms assume networks can be initialized so that for any input the output is zero. However, it is also assumed that any internal weights can easily be tuned by a training algorithm, such as gradient descent, making this initialization problem non-trivial.

To produce a constant zero output, it is possible to first initialize an arbitrary network, duplicate it, and subtract the outputs from each other. That is, given a function g(x) for some (randomly initialized) network, output g(x) - g(x) = 0.

We wrapped the logic for implementing this into something called an *R2R block*. Given some input, two convolutions are applied in parallel, with the same weight initialization W. After, they are passed through some per-channel function, in this case Batch Norm [2], followed by a ReLU activation function. Finally, 1x1 convolutions are applied to the two convolutions, this time with weight initializations β and -β. When summed together, the negation assures the output is zero. W and β can be initialized so that it is easily tuned.



To initialize a constant zero output from a network, one only needs to assure the output layer is appropriately initialized.

No symmetry breaking is required, as output layer symmetries can be broken via asymmetries in the loss function.

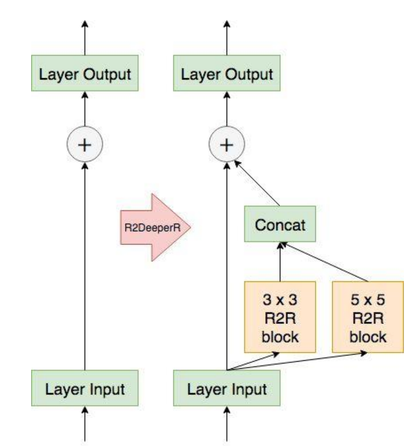
We verified our intuitions by training a "zero-initialized" network, with an R2R block for the output layer. We observed no negative impact on training and manually inspected weights to check that the symmetry was broken after a few training steps.

A practical implementation can provide the same effect without parallel convolutions, instead using concatenation.

Resnet2Resnet Transformations

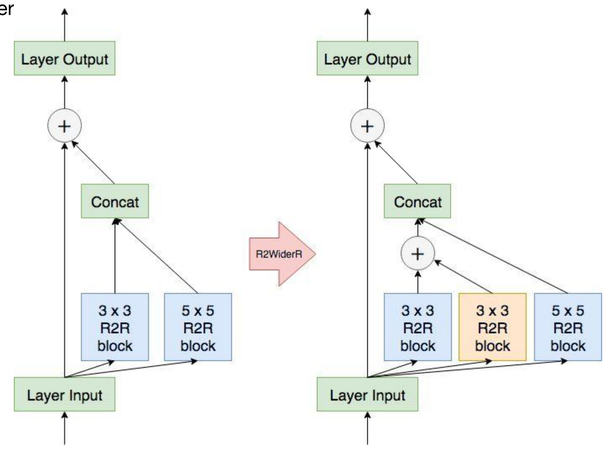
Resnet2Resnet (R2R) is the name given to our transformations, of which there are two: R2WiderR and R2DeeperR, that widen and deepen a network respectively.

R2DeeperR, depicted below, adds a new residual connection, and zero-initialized R2R blocks (orange), creating a new layer in the network. The output of the R2R blocks is zero, and therefore with the residual connection provides an identity transformation. Zero padding is added in the residual connection as required.



R2WiderR, also depicted below, adds new convolutions in parallel, in a layer that already exists. The new orange block is zero-initialized so that the output of the layer is not changed. The blue blocks contain the same weights before and after the transform.

Both transforms increase the capacity of a network while maintaining the same input/output characteristics as before. The new weights can then be fine-tuned by any training algorithm. Multiple transforms can be applied and combined at the same time, at any layer



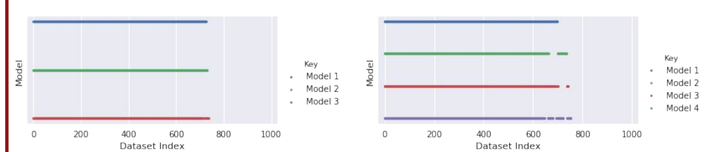
Neuroevolution

When R2R transformations are applied to a network, they do not alter any existing weights in the network, allowing the old and new network to co-exist. Applied repetitively, this can produce a *population* of networks, which share many parameters.

Topological neural network evolution (neuroevolution) has been used successfully in architecture searches [3], which automate architecture design. Additionally, function preserving transforms and parameter sharing have been used to efficiently evaluate architectures, in *efficient architecture searches* [4,5]. By using R2R transformations, we can combine these three ideas into a single architecture search algorithm.

Algorithm 1 (right) is pseudocode for our neuroevolution algorithm. To start, a single architecture h_i is trained for N_i steps. Each iteration consists of a selection stage, reducing the population of size M down to size K; a mutation stage, which applied R2R transforms randomly, to increase the population to a size of M; and a training stage, which picks N_j random models (with replacement) and performs a single update on each.

Note there is freedom in how to select models from the population, and to begin with, naively selecting the K best models on the validation set was used.



Algorithm 1 NE(select, mutate, N_i, N_j, K, M, h_i)

```
let modelSet = {h_i}
jointTrain(modelSet, N_i)                ▷ Trains just h_i.
for 1 to numIters do
    modelSet = select(modelSet, K)
    modelSet = mutate(modelSet, M)
    jointTrain(modelSet, N_j)
end for
return select(modelSet, K)
```

Diversity in the population can be encouraged by dynamically re-weighting examples when computing validation accuracy. After selecting the first model, any examples that have been correctly classified by a model already selected are given a lower weight. This encourages the population to "coevolve," and learn to classify collectively.

Left is two "classification profiles" which visualize which validation set examples are classified correctly by each model. The leftmost is the last three models when using greedy selection, and the right is the last four selected models when coevolution is used.

After coevolutionary training is performed, we can then train a hierarchical ensemble [6], to combine what was learned by all of the models into a single model.

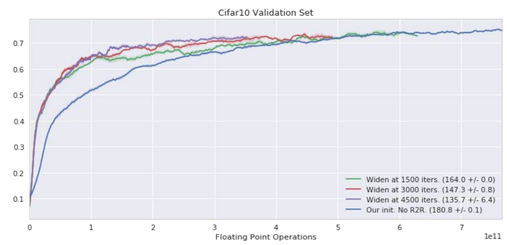
Results

Image classification on Cifar-10 was used as a simple task to evaluate R2R transformations. For all tests, our "base network" was a small network with 6 layers, and a fixed 3x3 kernel size.

When testing R2WiderR, initial training was performed on a network with half as many filters. Four training runs were performed, where at timesteps 0, 1500, 3000, 4500 R2WiderR was applied at every layer.

The graph below shows validation accuracy with respect to the number of floating point operations (flops) used for training so far. As a network with half as many filters requires fewer flops per update, the graphs end earlier the later the network was widened. When widening at 4500 iterations, the training run is 45 seconds quicker (a 25% speedup), with no impact in end performance.

Similar experiments were performed for R2DeeperR, with similar results, however, this is omitted for space.



To evaluate our neuroevolution work, we also used image classification on Cifar-10. To be convinced that neuroevolution had a positive impact on performance, the 6 layer base network (same as above) was trained for 10000 timesteps, beyond convergence, and evaluated on the test set every 1000 updates, to be unfair to the neuroevolution algorithms.

We used the following parameters in algorithm 1 (N_i, N_j, K, M) = (5000, 5000, 3, 10). The initial architecture, h_i, was three layers deep, using one 3x3 convolution per layer. Mutations we allowed to introduce 1x1, 3x3 and 5x5 R2R blocks. We report the following test accuracies: the network with best validation accuracy using just neuroevolution; the network with best validation accuracy using coevolution; and, the hierarchical ensemble trained on the final three models selected when using coevolution.

Model	Accuracy (Test set)
Base Model (best score)	0.70
Neuroevolution (best model)	0.74
Coevolution (best model)	0.72
Coevolution (ensemble)	0.75

References

[1] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. arXiv preprint arXiv:1511.05641, 2015.
[2] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.
[3] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. arXiv preprint arXiv:1703.01041, 2017.
[4] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Reinforcement learning for architecture search by network transformation. arXiv preprint arXiv:1707.04873, 2017.
[5] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268, 2018.
[6] https://blog.statsbot.co/ensemble-learning-d1dcd548e936