Evolving Modular Neural Sequence Architectures with Genetic Programming

David Dohan
Google Brain
ddohan@google.com

David So
Google Brain
davidso@google.com

Quoc Le
Google Brain
qvl@google.com

CCS CONCEPTS

• **Computing methodologies** → **Genetic programming**; **Neural networks**;

KEYWORDS

Genetic programming, sequence modeling, architecture search

ACM Reference Format:
David Dohan, David So, and Quoc Le. 2018. Evolving Modular Neural Sequence Architectures with Genetic Programming. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3205651.3208782

1 INTRODUCTION

Automated architecture search has demonstrated significant success for image data, where reinforcement learning and evolution approaches now outperform the best human designed networks ([12], [8]). These successes have not transferred over to models dealing with sequential data, such as in language modeling and translation tasks. While there have been several attempts to evolve improved recurrent cells for sequence data [7], none have achieved significant gains over the standard LSTM. Recent work has introduced high performing recurrent neural network alternatives, such as Transformer [11] and Wavenet [4], but these models are the result of manual human tuning.

In this work, we apply genetic programming techniques to search for improved non-recurrent sequence architectures specified in a tree-structured search space. We present the top architectures that were discovered and demonstrate their ability to outperform Transformer models, transfer across tasks, and trade off between speed and accuracy.

2 SEARCH SPACE

2.1 Motivation

Defining the search space is a core aspect of architecture search. We wanted to create a space that was flexible enough to express most state of the art models, while also behaving nicely for our search algorithm, evolution. To address the first requirement, we composed the search space vocabulary of fundamental operations and functions (Table 1). These are the low level building blocks of many effective neural networks, and are sufficient to reproduce our baseline model, the Transformer.

Table 1: Operators in the search space.

Operation	Description
Identity(x)	x
Add(x, y)	$x + y$
Mul(x, y)	$x \odot y$
Gate(x, y)	$x \odot \sigma(y)$
Conv(x)	Convolution with variable kernel width, dilation rate (DConv), and separability (SConv).
SelfAttention(x)	Apply self attention.
LocalAttention(x)	Apply self attention with limited receptive field.
Nonlinearity(x)	One of [Relu, σ, Tanh, Swish]
LayerNorm(x)	Layer normalization
Sequential(f, g, x)	$f(g(x))$
Parallel(f, g, h, x)	$f(g(x), h(x))$
ParallelClone(f, g, x)	$f(g(x), g(x))$
Repeat2(f, x)	$f(f(x))$
ResidualNorm(f, x)	$f(\text{Layernorm}(x)) + x$
Highway(h, t, c, x) [9]	$\sigma(t(x)) \odot h(x) + \sigma(c(x)) \odot x$
CoupledHighway(h, t, x)	$\sigma(t(x)) \odot h(x) + (1 - \sigma(t(x))) \odot x$

To address the second requirement, we compose these operations into tree structures, where the space of valid trees is defined by a context free grammar (CFG). In practice, this is similar to the graph grammars discussed by [2] and defines an indirect encoding mechanism. The field of NeuroEvolution has many alternative graph-based techniques for searching over graphs in a direct encoding, such as Cartesian Genetic Programming [3] and NEAT [10]. We choose the tree representation in order to make the modularity of the underlying networks explicit, allowing effective subtrees to be reused multiple times by higher order functions.

2.2 Language Overview

All operations in the space are either unary or binary. Primitive operations map $tensor \rightarrow tensor$, while higher order operations accept operations as arguments and return a function that map $tensor \rightarrow tensor$.

A simple example demonstrating how one would encode a stack of two residual blocks is:

$$\text{Repeat}(\text{Parallel}(f{=}\text{Add}, g{=}\text{Identity}, h{=}\text{Sequential}(\text{Relu}, \text{Conv})))$$

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07.
https://doi.org/10.1145/3205651.3208782

Architecture	WMT	CIFAR	WMT Steps/Sec	CIFAR Steps/Sec
Transformer (Baseline): Repeat6(Sequential(ResidualWithNorm(SelfAttention), ResidualWithNorm(ProjectUp(Relu))))	2.128	2.630	1.81	5.10
Image Transformer (Baseline): Repeat6(Sequential(ResidualWithNorm(LocalAttention), ResidualWithNorm(ProjectUp(Relu))))	3.049	2.477	7.90	4.78
Sequential(LayerNorm, Conv3)	2.151	3.914	**15.65**	**5.90**
ResidualNorm(CoupledHighway(h=Repeat4(CoupledHighway(h=SConv7, t=DSConv7)), t=Sigmoid))	**2.068**	2.974	8.768	5.25
CoupledHighway(..., ResidualNorm(CoupledHighway(h=DSConv3, t=DSConv7))[1]	2.191	**2.372**	3.91	3.606

Table 2: Comparison of a few top architectures across translation (WMT) and image generation (CIFAR). Both datasets are evaluated in terms of the log-likelihood[2] of the dev set under the model (lower is better). Results are reported at 60k steps for WMT and 1 epoch (50k steps) for CIFAR.

3 SEARCH METHOD

For a search with N workers, the population is initialized with N random architectures generated using ramped half-and-half initialization. Evolution occurs asynchronously, with workers requesting models on an as-needed basis, as different models may have different run times. When a new model is requested, a parent is selected from the existing population via a tournament selection, wherein 5% of the population of trained models is randomly sampled and the best one is used as the parent. With 10% probability, another parent is drawn for single point crossover, otherwise a single mutation is applied. Individuals are never removed from the population

For crossover, a node of the same parity is chosen in each tree and swapped. For mutation, a randomly chosen node is either mutated to another node of the same type or replaced with a randomly generated subtree. We implement our search using the Distributed Evolutionary Algorithms in Python library [1].

4 EXPERIMENTS

We use the Transformer [11] and Image Transformer [6] models inside the Tensor2Tensor framework as baselines.

During the search, models are trained for a fixed step or time budget. When a search is done, we run the top models without a fixed step count. We show results comparing the best models discovered in the search to the Transformer baselines in tables 2 in terms of speed and log likelihood of the data. The top models in each case are drastically different architecture wise, but perform comparably or better than the baselines with equal parameter caps. The hidden size of each model is adjusted to achieve a total of 20 million parameters to make results comparable. Dev set performance is used as the reward during the search.

4.1 WMT English-German Translation

We train each model for 60k steps as it was empirically found to be a good point to achieve 0.9 Pearson correlation to final performance while minimizing training time (each model takes about 4 hours on an NVIDIA P100 GPU).

We restrict our search to the encoder of the standard encoder-decoder Transformer model. The decoder is fixed to be the standard Transformer to generate the output text, and the input text is processed by the generated architecture. This is one reason a simple `Conv3(LayerNorm(x))` does so well in this use case (with a hidden size of 2416 to meet the 20 million parameter budget). The decoder can connect longer term dependencies without the encoder.

[1]Image generation results are often reported in bits per dimension (bpd), which is the log-likelihood base 2. We report in base e

[2]Details omitted for brevity. The best performing strings often bloat in size since there is no length penalty.

4.2 CIFAR Image Generation

Image generation aims to find a model that can assign a probability an image and generate new ones conditional on known information [5]. Images are unraveled to a sequence and generated pixel-by-pixel, with performance measured in terms of bits per dimension, or $\log_2$ likelihood. The generation of pixel T is conditioned on the previous pixels $1 \ldots (T-1)$.

5 CONCLUSION AND FUTURE WORK

We have outlined a language and method for searching over architectures using genetic programming. The procedure uncovers architectures that perform on par with a Transformer baselines, while bearing little resemblance to current state of the art architectures. This work assumes that tensors share the same rank and length. This can be addressed by extending the language to allow higher level typed operators, such as defining a function on a single vector then mapping that function over a tensor. We believe this is the path to discovering novel and interesting network building blocks going forward. There are also opportunities to improve the search space and controller, including adding automatically defined functions to allow reusing a block, and search additions such as niching or a novelty reward to prevent the population stagnation.

REFERENCES

[1] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.

[2] Frederic Gruau et al. 1994. Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm. (1994).

[3] Julian F Miller. 2011. Cartesian genetic programming. In *Cartesian Genetic Programming*. Springer, 17–34.

[4] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[5] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. 2016. Conditional image generation with pixelcnn decoders. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Curran Associates Inc., 4797–4805.

[6] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, and Alexander Ku. 2018. Image Transformer. *arXiv preprint arXiv:1802.05751* (2018).

[7] A. Rawal and R. Miikkulainen. 2018. From Nodes to Networks: Evolving Recurrent Neural Networks. *ArXiv e-prints* (March 2018). arXiv:1803.04439

[8] E. Real, A. Aggarwal, Y. Huang, and Q. V Le. 2018. Regularized Evolution for Image Classifier Architecture Search. *ArXiv e-prints* (Feb. 2018). arXiv:1802.01548

[9] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. *arXiv preprint arXiv:1505.00387* (2015).

[10] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 6000–6010.

[12] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2017. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012* (2017).