NeuralArTS: Structuring Neural Architecture Search with Type Theory

Robert Wu Nayan Saxena Rohan Jain

University of Toronto ML Collective

Introduction

Neural Architecture Search (NAS) has proven to be a complex but important area of deep learning research. The aim of NAS is to automatically design architectures for neural networks. Most NAS frameworks involve sampling operations from a search space. For example, Efficient Neural Architecture Search (ENAS) uses a controller based on reinforcement learning (RL) to sample child networks Pham et al. [2018]. Yu et al. [2019] and other recent works have identified flaws in NAS algorithms, motivating improved methods for network construction. Search spaces are usually manually developed pre-search, rely heavily on researchers' domain knowledge, and often involve trial and error; it's more of an art than a science. Additionally, the domain of network operations is infinite given the multitude of basic operations and hyperparameters therein. It's hard to know which operations produce better performance in learning tasks. In this paper, we introduce a framework to potentially improve search spaces using generation and heuristics.

Neural Architecture Type System (NeuralArTS)

Artificial neural networks can be interpreted as a programming domain, where operations can be categorized into type systems. A type system $\mathcal{T}$ is a formal system in which every element has a *type* τ, which defines its meaning and the operations that may be performed on it [Coquand, 2018]. One intuitive property of networks is the shape of the data as it moves through layers. Classes of operations such as pooling or convolution layers have mappings between input/output (I/O) dimensions, which can range from totally flexible (can be placed anywhere in a network) to complex (perhaps requiring a specific input and output shape). Therefore, it is possible to categorize operations into a type system based on these I/O mappings, opening new possibilities for search space optimisation in NAS. This idea can be extended to network subgraphs, since they can be abstracted as a block of operations with compound dimension functions. A consequence is that layers and subgraphs are also interoperable.

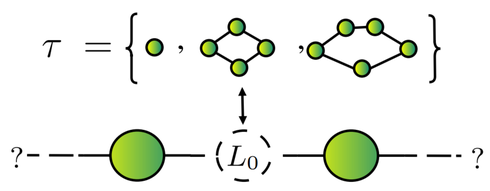

Figure 1. Possible interchange of $L_0 \in \tau$ with other $\tilde{L} \in \tau$ or subgraph $[\tilde{L}_1, \ldots, \tilde{L}_n] \in \tau$. The unspecified topologies on either end represent previous/subsequent layers.

This idea is intuitive and not entirely new; it's explored in Elsken et al. [2021], albeit informally. To be formal and precise, the domain $\mathcal{U}$ of network operations can be categorized with a type system $\mathcal{T}$ centred around data shape compatibility. Compatibility is fundamental in informing which operations can precede, follow, or replace each other. For each operation layer L with shape dimensions $\{1, \ldots, d\}$, define $I_L = (I_L^{(1)}, \ldots, I_L^{(d)})$ and $O_L = (O_L^{(1)}, \ldots, O_L^{(d)})$ to be the shapes of I/O data. Assume I/O shapes have the same number of dimensions for simplicity.

Dimension Functions

While these shapes can be constants, they are generally mappings that can be defined as a *dimension function* f_L. Operation layer L can have several other properties such as depth, stride, or bias values. Some or all of these properties may influence O_L, and they can be encapsulated in f_L. A dimension function can therefore be abstracted as $O_L := f_L(I_L)$.

Equivalence Properties

NeuralArTS can centre around replacement/interchange of operation layers. Let $L_A, L_B \in \mathcal{U}$ be arbitrary layers with I/O dimensions (I_{L_A}, O_{L_A}) and (I_{L_B}, O_{L_B}) respectively.

Complete-Equivalence L_A and L_B are *completely equivalent* if all of their properties are equivalent. $L_A = L_B \iff (I_{L_A}, \ldots, O_{L_A}) = (I_{L_A}, \ldots, O_{L_B})$

Type-Equivalence L_A and L_B are considered *type-equivalent* if their I/O dimension functions are equivalent. In other words, L_A and L_B belong to the same type, τ. $L_A \sim L_B \iff f_{L_A} = f_{L_B}$

Instant-Equivalence L_A and L_B are *instant-equivalent* at input size I if their I/O dimension functions intersect at I. $L_A \perp_I L_B \iff f_{L_A}(I) = f_{L_B}(I)$

Sequential Compatibility

Let $\mathcal{I}_L$ be the domain of acceptable input shapes, and $\mathcal{O}_L$ be the range of output shapes.

Forward-Compatibility L_A is *forward compatible* to L_B if all output shapes of L_A are acceptable as input to L_B. $L_A \to L_B \iff \mathcal{O}_{L_A} \subseteq \mathcal{I}_{L_B}$

Complete-Compatibility L_A is *completely compatible* to L_B if they're mutually forward-compatible. $L_A \leftrightarrow L_B \iff L_A \to L_B \wedge L_B \to L_A$

These properties regarding sequential compatibility of L_A and L_B are potentially useful in network construction; the controller in ENAS can change to consider compatibility in making direct and skip connections [Pham et al., 2018].

Example: Convolutional Layers

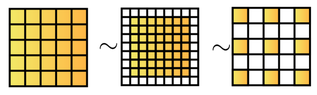

Figure 2. Type-equivalent convolutions that can be interchanged: a 5x5; a 7x7 with $p = 1$; and a 3x3 with $d = 2$.

Let $\mathcal{S}_0$ be the original ENAS search space [Pham et al., 2018] and $C \in \mathcal{S}_0$ be a convolution with kernel size k, padding p, and dilation d. These properties dictate type-equivalent convolutions. To test the efficacy of NeuralArTS, we first added some dilated variants of convolutions to $\mathcal{S}_0$. We found that adding even a single such convolution can outperform the baseline (3E in Figure 3).

$$f_C^{(i)}(I_C^{(i)}) := \left\lfloor \frac{I_C^{(i)} + 2p_C^{(i)} - d_C^{(i)}(k_C^{(i)} - 1) - 1}{s_C^{(i)}} \right\rfloor + 1 \tag{1}$$

Generating Algorithm

We introduce a generation technique that, without loss of generality, bounds two of (k', p', d') to (K, P, D) and derives the third using the convolution's I/O dimension function $f_C^{(i)}(I_C^{(i)})$ detailed in equation 1. Let (k, p, d) be the properties of the original seed operation $L_0 \in \tau$. Let (K, P, D) be the generation parameters, which include exactly one `None` and two positive integer ranges (inclusive). The algorithm in Figure 4 explores the Cartesian product of (K, P, D) to produce candidate tuples (k', p', d') of properties. Each tuple's `None` value is replaced and derived from the other two values and the original properties (k, p, d).

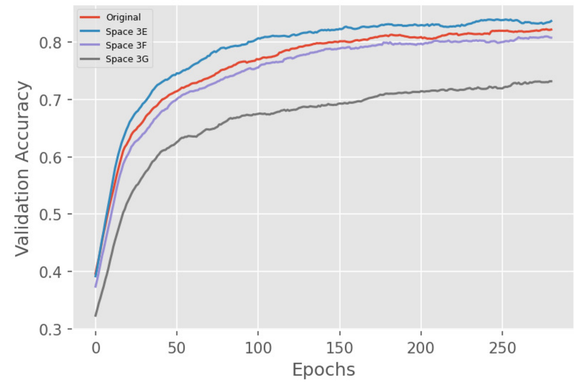

Figure 3. Validation accuracy graphed over time of ENAS on various search spaces with added dilated convolutions. ENAS baseline is labelled "Original" in with a red graph. Note that 3E outperforms baseline in validation accuracy.

Algorithm 1: GenerateTypeEquivalentConvs

let $settings = []$
assert (K, P, D).count(None) $= 1$
for $(k', p', d') \in (K \times P \times D)$ **do**
  if $K =$ None **then derive** $k' = \frac{2p' - 2p - d(k-1)}{d'+1}$
  else if $P =$ None **derive** $p' = \frac{d'(k'-1)}{2}$
  else if $D =$ None **derive** $d' = \frac{2p'}{k'-1}$
  if $k', p', d' \in \mathbb{N}$ **then** $settings$.append$((k', p', d'))$
end for
return settings

Figure 4. Algorithm to generate type-equivalent convolutions given two of {kernel size, padding, dilation} ranges.

Conclusion & Directions for Future Work

This generation method can be improved with "smarter" domains of operation properties; if performance proves to be continuous with respect to these properties, linear or manifold optimization might help generate more refined search spaces to speed up NAS. Another exciting prospect is that NeuralArTS can act as a heuristic for pre-optimized search spaces. It can naïvely eliminate completely-equivalent (or even type-equivalent) operations in preprocessing. More practical is changing the controller to modulate operation likelihoods at the type (rather than operation) level. If performance for types could be generalized, NeuralArTS can also be used to hierarchize NAS by performing shallow type-searches first, and then choosing random or "best" representative(s) from each type. We hypothesize these directions might lead to improvements of NAS algorithms.

References

Thierry Coquand. Type theory. *The Stanford Encyclopedia of Philosophy (Fall 2018 Edition), Edward N. Zalta (ed.)*, 2018.

Thomas Elsken, Benedikt Staffler, Arber Zela, Jan Hendrik Metzen, and Frank Hutter. Bag of tricks for neural architecture search. *arXiv preprint arXiv:2107.03719*, 2021.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.