NeuralArTS: Structuring Neural Architecture Search with Type Theory

Robert (Rupert) Wu, Nayan Saxena, & Rohan Jain

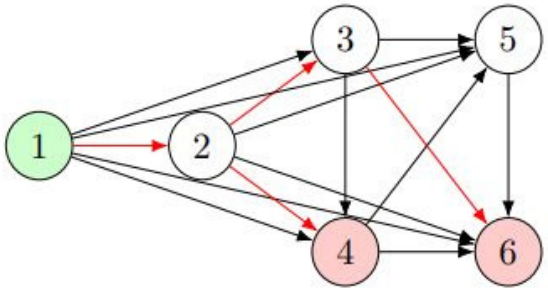
University of Toronto

https://www.cs.toronto.edu/~rupert

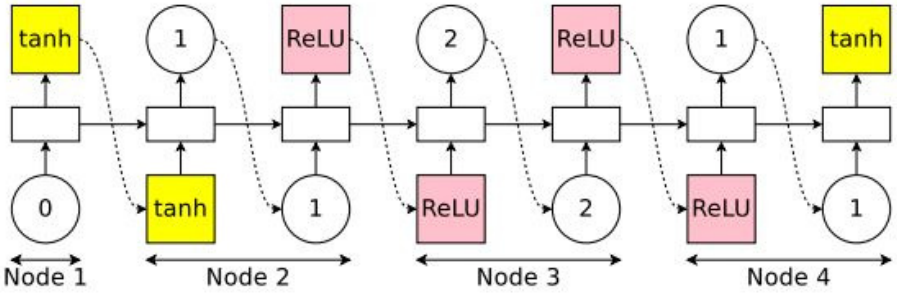
January 31, 2022

Efficient Neural Architecture Search (ENAS)

- **Search Space:** Represented as DAG, each node being one operation.



- **Search Strategy:** LSTM controller samples layers for child networks.



- **Performance Estimation:** Child model shared parameters are trained using cross entropy loss and parameters for LSTM controller trained with REINFORCE (Williams, 1992). Goal = Maximize Reward (Validation Accuracy)

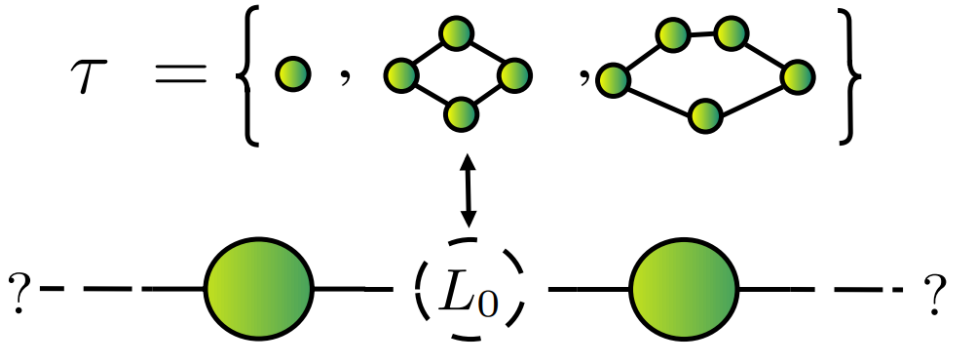
Search Spaces & Type Theory

- NAS algorithms are generally expensive due to the magnitude of the search space.
- A type system $\mathcal{T}$ is a formal system in which every element $L \in \mathcal{T}$ has a *type* τ, which defines its meaning and operations.

$$\forall L \in \mathcal{T}, \exists \tau \in \mathcal{T}, L \in \tau$$

- Shapes of data! Like input/output.
- Categorize them in a type system.
- Make guarantees or identify redundancies to optimize search spaces.
- Could extend to subgraphs to make them interoperable to subgraphs.

Interoperability



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.

Shape of Data

Let I_L and O_L be the input/output (I/O) shapes of an operation layer L. While these shapes can be constants, they are generally mappings that can be defined as a *dimension function* f_L of I_L.

$$O_L := f_L(I_L)$$

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.

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.

Definition (Complete-Equivalence)

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

Definition (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. $L_A \sim L_B \iff f_{L_A} = f_{L_B}$

Definition (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 into L, and $\mathcal{O}_L$ be the range of output shapes produced from L.

Definition (Forward-Compatibility)

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

Definition (Complete-Compatibility)

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

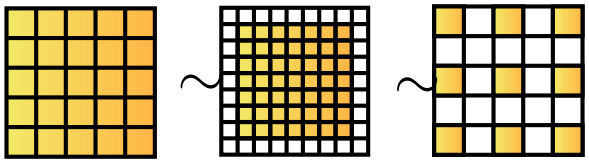
These properties regarding sequential compatibility of L_A and L_B are potentially useful in network construction; the controller in ENAS can change to consider compatibility in making direct and skip connections.

Generative Example: Convolutional Layers

Convolutional layers (perhaps in a CNN) have the following dimension function as a function of kernel k, padding p, dilation d, and stride s.

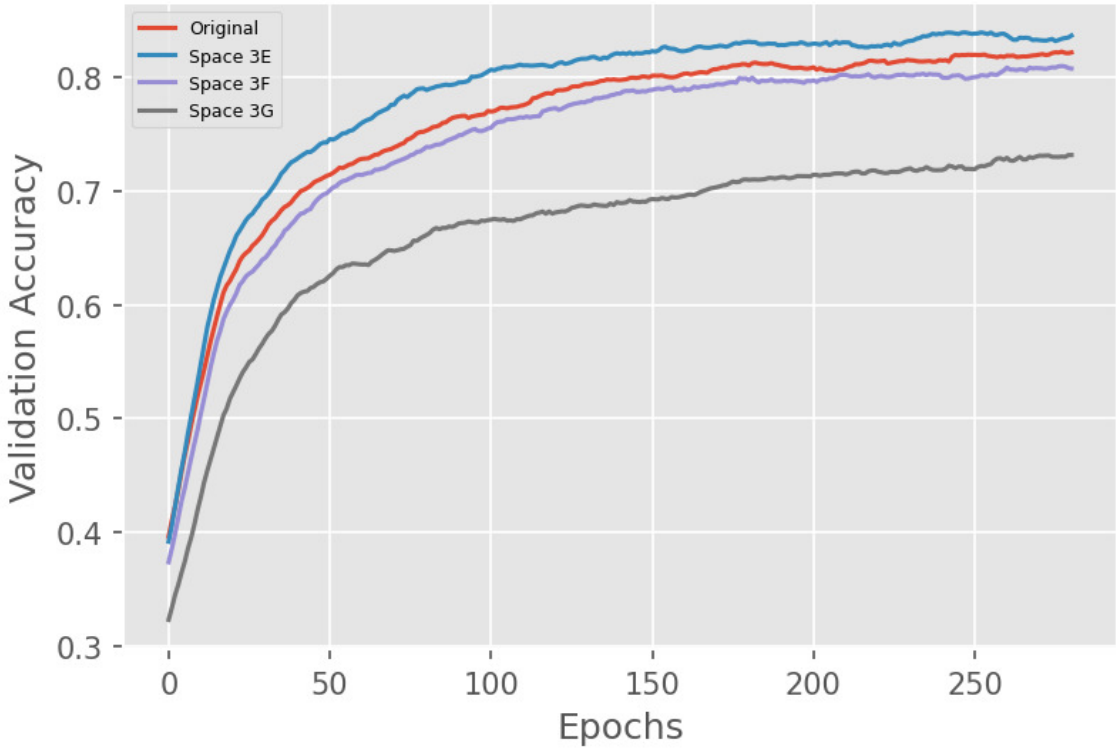
$$f_c^{(i)}(I_c^{(i)}) := \left\lfloor \frac{I_c^{(i)} + 2p_c^{(i)} - d_c^{(i)}(k_c^{(i)} - 1) - 1}{s_c^{(i)}} \right\rfloor + 1$$

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



Preliminary Experiments

- Let $\mathcal{S}_0$ be the original ENAS baseline search space.
- Adding a type-equivalent convolution $C \in \mathcal{S}_0$ can improve it.



Search Space	Val Acc	Test Acc
S_0 (Baseline)	80.47%	74.67%
$S_0 + \text{Conv}(k = 3, p = 2, d = 2)$	**81.91%**	**78.64%**

A Generating Algorithm

We should therefore be able to generate more operations for a search space! Redundancies can likewise be identified.

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

Directions for Future Work

This generation method can be improved with "smarter" domains; if performance estimation is continuous, search spaces can be automatically optimized. Furthermore, *NeuralArTS* can...

- Serve as heuristic for pre-optimized search spaces.
- Eliminate redundant type-equivalent operations.
- Hierarchize NAS by performing shallow type-searches first, and then choosing random or "best" representative(s) from each type. In other words, "explore first, exploit later".