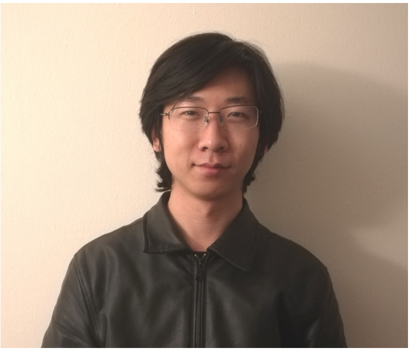
RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

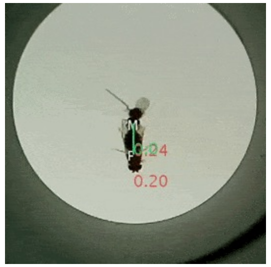
Differentiable Synthesis of Program Architectures



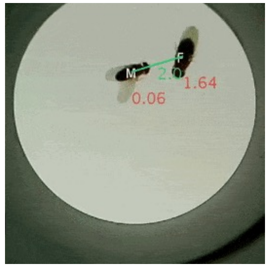
Guofeng Cui

He Zhu

Sequence Classification Tasks



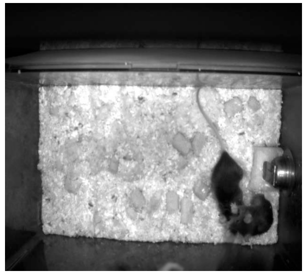
Collision



Wing Extension

Fly-vs-fly Dataset

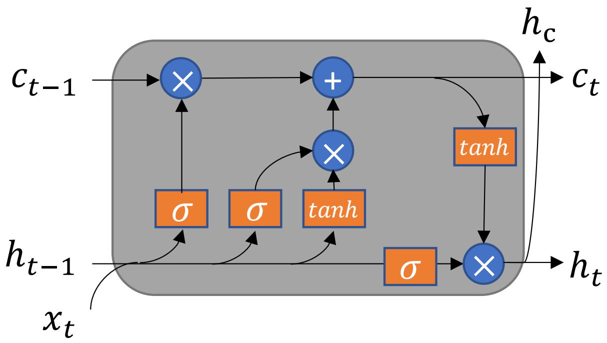
Walk Away



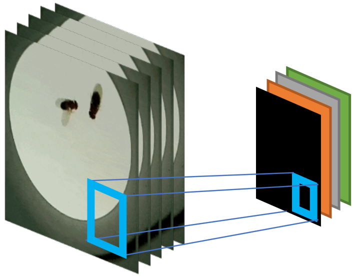
Sniff

Crim13 Dataset

- Classify a sequence input to a certain category



LSTM



CNN

Model is not easily interpretable!

Interpretable Sequence Classification via Program Synthesis

- Define a *Context-free DSL Grammar* to define a program architecture search space.

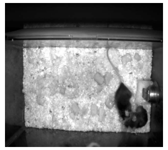
$$\alpha ::= x \mid c \mid \mathbf{Add}\ \alpha_1\ \alpha_2 \mid \mathbf{Multiply}\ \alpha_1\ \alpha_2 \mid \mathbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3 \mid \mathbf{F}_{\mathbf{S},\theta}(x) \mid \mathbf{map}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid \mathbf{mapprefix}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid \mathbf{fold}(\mathbf{fun}\ x_1.\alpha_1)\ c\ x \mid \mathbf{SlideWindowAvg}\ (\mathbf{fun}\ x_1.\alpha_1)\ x$$

Interpretable Sequence Classification via Program Synthesis

- Define a *Context-free DSL Grammar* to define a program architecture search space.

$$\alpha \ ::= x \mid c \mid \mathbf{Add}\ \alpha_1\ \alpha_2 \mid \mathbf{Multiply}\ \alpha_1\ \alpha_2 \mid \mathbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3 \ \Big|\ \mathbf{F}_{\mathbf{S},\theta}(x) \ \Big|\ \mathbf{map}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid \mathbf{mapprefix}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid \mathbf{fold}(\mathbf{fun}\ x_1.\alpha_1)\ c\ x \mid \mathbf{SlideWindowAvg}\ (\mathbf{fun}\ x_1.\alpha_1)\ x$$

x: program input as a sequence of frames where each frame contain a set of features

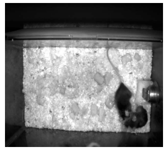
{ features(), features(), …. }

Interpretable Sequence Classification via Program Synthesis

- Define a *Context-free DSL Grammar* to define a program architecture search space.

$$\alpha \ ::= x \mid c \mid \mathbf{Add}\ \alpha_1\ \alpha_2 \mid \mathbf{Multiply}\ \alpha_1\ \alpha_2 \mid \mathbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3 \mid \mathbf{F}_{\mathbf{S},\theta}(x) \mid \mathbf{map}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid$$
$$\mathbf{mapprefix}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid \mathbf{fold}(\mathbf{fun}\ x_1.\alpha_1)\ c\ x \mid \mathbf{SlideWindowAvg}\ (\mathbf{fun}\ x_1.\alpha_1)\ x$$

x: program input as a sequence of frames where each frame contain a set of features

{ features(), features(), …. }

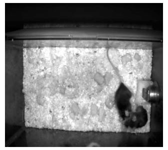
c, $\mathbf{Add}$, $\mathbf{Multiply}$: constant, arithmetic operations

Interpretable Sequence Classification via Program Synthesis

- Define a *Context-free DSL Grammar* to define a program architecture search space.

$$\alpha \ ::= x \mid c \mid \textbf{Add}\ \alpha_1\ \alpha_2 \mid \textbf{Multiply}\ \alpha_1\ \alpha_2 \mid \textbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3 \mid \mathbf{F}_{\mathbf{S},\theta}(x) \mid \textbf{map}\ (\textbf{fun}\ x_1.\ \alpha_1)\ x \mid \textbf{mapprefix}\ (\textbf{fun}\ x_1.\ \alpha_1)\ x \mid \textbf{fold}(\textbf{fun}\ x_1.\ \alpha_1)\ c\ x \mid \textbf{SlideWindowAvg}\ (\textbf{fun}\ x_1.\ \alpha_1)\ x$$

x: program input as a sequence of frames where each frame contain a set of features

{ features(), features(), …. }

c, $\textbf{Add}$, $\textbf{Multiply}$: constant, arithmetic operations

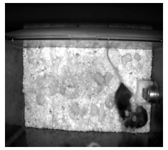
$\textbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3$: If-Then-Else

Interpretable Sequence Classification via Program Synthesis

- Define a *Context-free DSL Grammar* to define a program architecture search space.

$$\alpha ::= x \mid c \mid \mathbf{Add}\ \alpha_1\ \alpha_2 \mid \mathbf{Multiply}\ \alpha_1\ \alpha_2 \mid \mathbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3 \mid \mathbf{F}_{\mathbf{S},\theta}(x) \mid \mathbf{map}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid \mathbf{mapprefix}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid \mathbf{fold}(\mathbf{fun}\ x_1.\alpha_1)\ c\ x \mid \mathbf{SlideWindowAvg}\ (\mathbf{fun}\ x_1.\alpha_1)\ x$$

x: program input as a sequence of frames where each frame contains a set of features

{ features(), features(), …. }

c, $\mathbf{Add}$, $\mathbf{Multiply}$: constant, arithmetic operations

$\mathbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3$: If-Then-Else

$\mathbf{F}_{\mathbf{S},\theta}(x)$: parameterized function that extracts a subset $\mathbf{S}$ of features from a data frame x and passes the extracted features through a linear function with parameters θ (for interpretability)

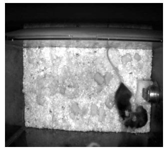
$\mathbf{F}_{\mathbf{S},\theta}(x)$	S	Semantics
PositionAffine	0, 1, 2, 3	mice positions
DistanceAffine	4	mice distance

Interpretable Sequence Classification via Program Synthesis

- Define a *Context-free DSL Grammar* to define a program architecture search space.

$$\alpha \ ::= x \mid c \mid \mathbf{Add}\ \alpha_1\ \alpha_2 \mid \mathbf{Multiply}\ \alpha_1\ \alpha_2 \mid \mathbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3 \mid \mathbf{F}_{\mathbf{S},\theta}(x) \mid \mathbf{map}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid$$
$$\mathbf{mapprefix}\ (\mathbf{fun}\ x_1.\alpha_1)\ x \mid \mathbf{fold}(\mathbf{fun}\ x_1.\alpha_1)\ c\ x \mid \mathbf{SlideWindowAvg}\ (\mathbf{fun}\ x_1.\alpha_1)\ x$$

x: program input as a sequence of frames where each frame contains a set of features

{ features(), features(), …. }

c, $\mathbf{Add}$, $\mathbf{Multiply}$: constant, arithmetic operations

$\mathbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3$: If-Then-Else

$\mathbf{F}_{\mathbf{S},\theta}(x)$: parameterized function that extracts a subset $\mathbf{S}$ of features from a data frame x and passes the extracted features through a linear function with parameters θ (for interpretability)

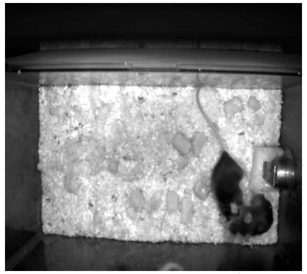
$\mathbf{F}_{\mathbf{S},\theta}(x)$	S	Semantics
PositionAffine	0, 1, 2, 3	mice positions
DistanceAffine	4	mice distance

$\mathbf{map}$, $\mathbf{mapprefix}$, $\mathbf{fold}$, $\mathbf{SlideWindowAvg}$: standard higher-order combinators to recurse over sequences

Interpretable Sequence Classification via Program Synthesis

- Synthesize a program in the DSL to classify a sequence of actions made by two mice to "sniff" or "no sniff".

Walk Away



Sniff

Position bias

Map(
  Multiply(
    PositionAffine $_{\theta_1}(x_t)$, DistanceAffine $_{\theta_2}(x_t)$)) x

Doing "sniff" if two mice are close

Crim13 Dataset

Program Synthesized

- Problem Formulation:

$$\alpha \ ::= x \mid c \mid \textbf{Add}\ \alpha_1\ \alpha_2 \mid \textbf{Multiply}\ \alpha_1\ \alpha_2 \mid \textbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3 \mid \textbf{F}_{\textbf{S},\theta}(x) \mid \textbf{map}\ (\textbf{fun}\ x_1 . \alpha_1)\ x \mid \textbf{mapprefix}\ (\textbf{fun}\ x_1 . \alpha_1)\ x \mid \textbf{fold}(\textbf{fun}\ x_1 . \alpha_1)\ c\ x \mid \textbf{SlideWindowAvg}\ (\textbf{fun}\ x_1 . \alpha_1)\ x$$

$$\arg\min_{\alpha,\theta} \mathcal{L}(P(\cdot;\ \alpha,\theta)) \text{ where } \mathcal{L}(P(\cdot;\ \alpha,\theta)) = \mathbb{E}_{i_k,o_k\sim D}[\ell(P(i_k;\ \alpha,\theta),o_k)]$$

synthesize a program P

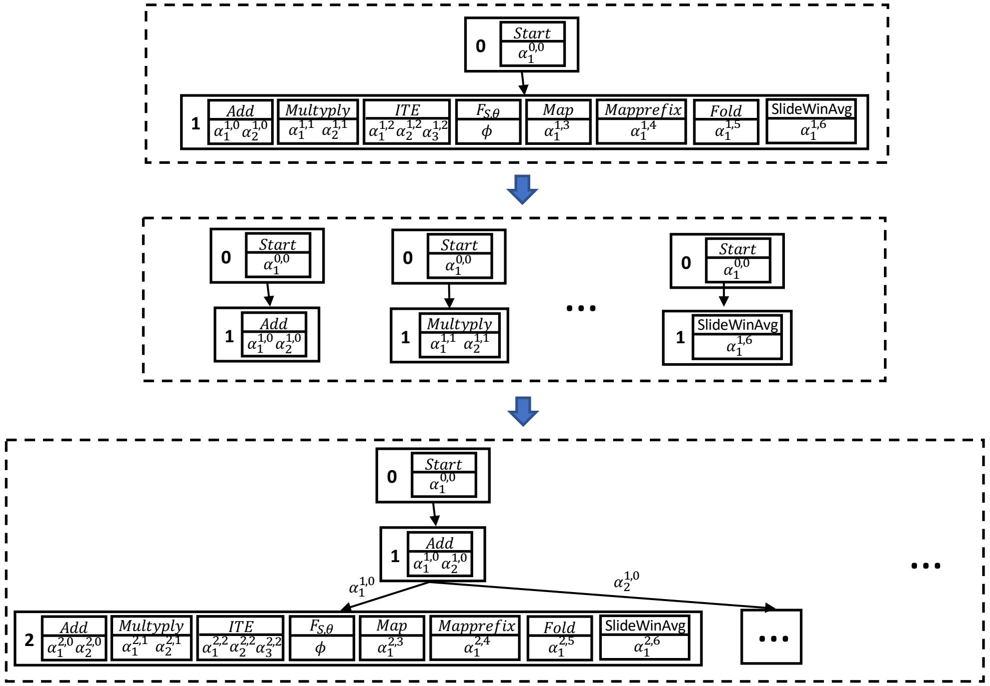
prediction error loss on a sequence i_k w.r.t its category o_k

Interpretable Sequence Classification via Program Synthesis

- Define a *Context-free DSL Grammar* to define a program architecture search space.

$$\alpha \ ::= x \mid c \mid \textbf{Add}\ \alpha_1\ \alpha_2 \mid \textbf{Multiply}\ \alpha_1\ \alpha_2 \mid \textbf{ITE}\ \alpha_1 \geq 0\ \ \alpha_2\ \alpha_3 \mid \mathbf{F}_{\mathbf{S},\theta}(x) \mid \textbf{map}\ (\textbf{fun}\ x_1 .\ \alpha_1)\ x \mid \textbf{mapprefix}\ (\textbf{fun}\ x_1 .\ \alpha_1)\ x \mid \textbf{fold}(\textbf{fun}\ x_1 .\ \alpha_1)\ c\ x \mid \textbf{SlideWindowAvg}\ (\textbf{fun}\ x_1 .\ \alpha_1)\ x$$

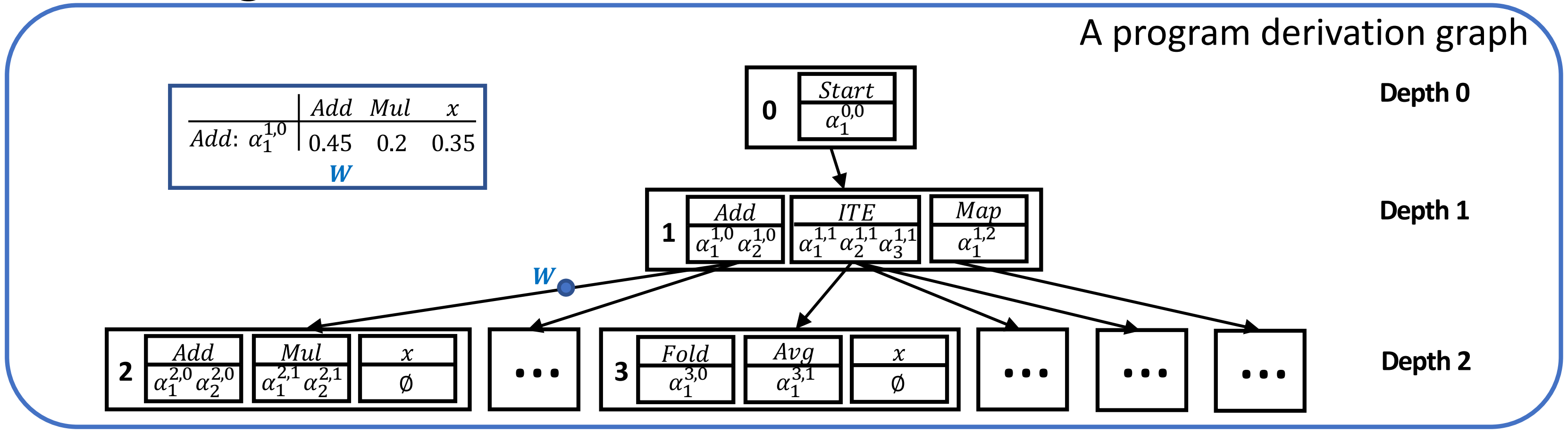
- ***Challenge*** - Discrete and combinatorial search for programmatic classifiers.



Architecture Enumeration is Inefficient!

Differentiable Program Architecture Synthesis

- Differentiable Program Derivations.



  - Encode the entire search space (up to a depth bound) as a differentiable program $T_{w,\theta}$ with architecture weight w and the parameters θ in all programs sharing the search space.
  - Program synthesis as optimizing $T_{w,\theta}$ with respect to the accuracy loss on training examples.
    - w and θ learned via *bi-level optimization* using gradient descent.

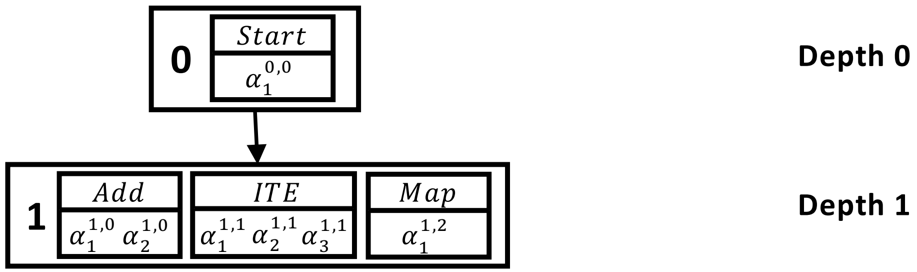
- Differentiable Program Semantics.

$$[\![\textbf{ITE}\ (\alpha_1 \geq 0, \alpha_2, \alpha_3)]\!](x) = \sigma\big([\![\alpha_1]\!](x)\big) \cdot [\![\alpha_2]\!](x) + \big(1 - \sigma[\![\alpha_1]\!](x)\big) \cdot [\![\alpha_3]\!](x)$$

However, training is still difficult as $T_{w,\theta}$ is exponentially large!

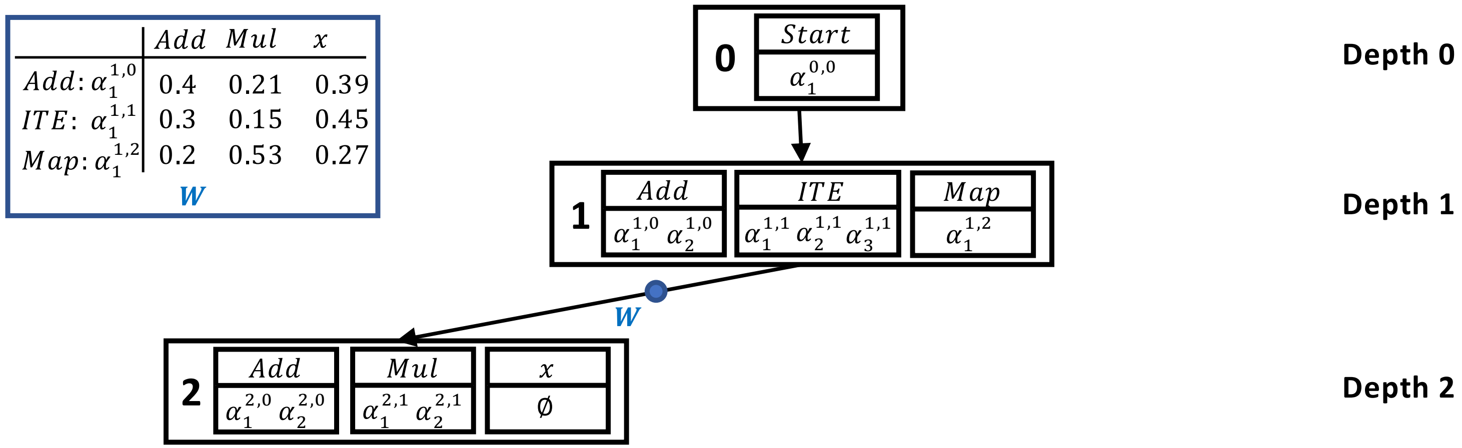
Optimizing Differentiable Architecture Search

- 1. ***Node Sharing***.



Optimizing Differentiable Architecture Search

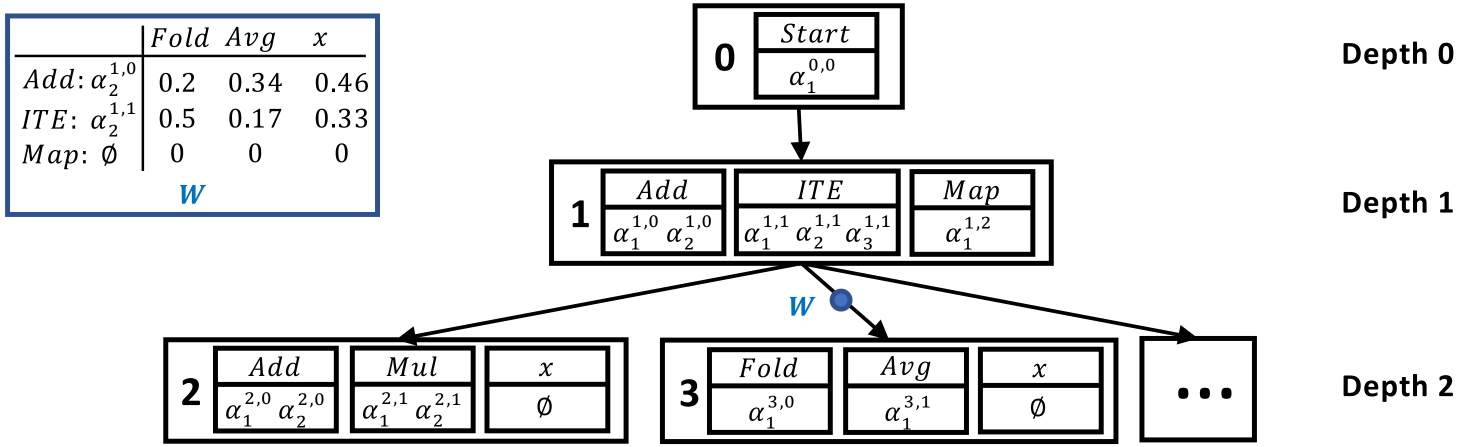
- 1. ***Node Sharing***.



- Nonterminals in partial architectures of the same node ***share*** child nodes.
  - For example, the first parameters of Add $\alpha_1^{1,0}$ and ITE $\alpha_1^{1,1}$ on node 1 share child node 2.

Optimizing Differentiable Architecture Search

- 1. ***Node Sharing***.



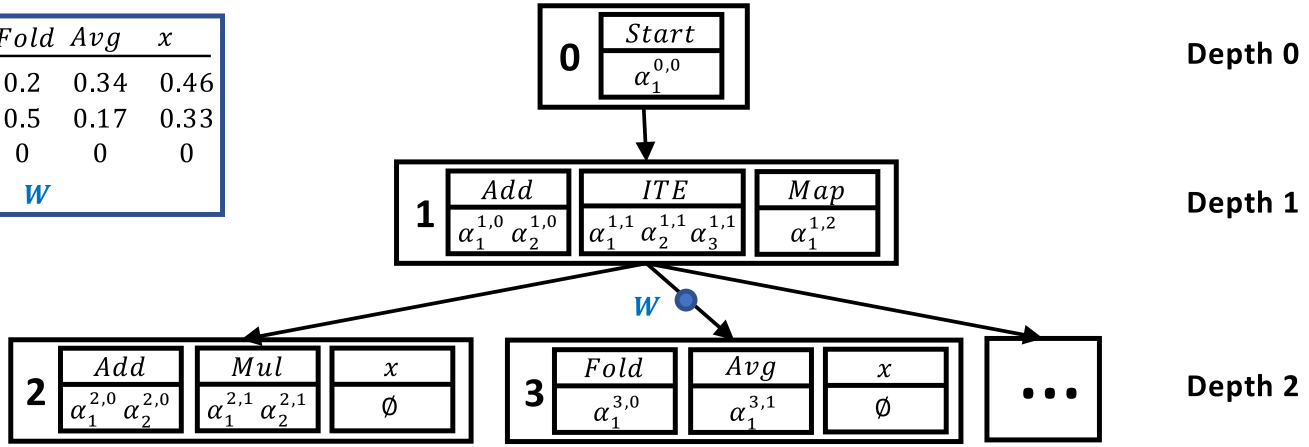
- Nonterminals in partial architectures of the same node ***share*** child nodes.
  - For example, the second parameters of Add $\alpha_2^{1,0}$ and ITE $\alpha_2^{1,1}$ on node 1 share child node 3.

Optimizing Differentiable Architecture Search

- 1. ***Node Sharing***.

	Fold	*Avg*	*x*
$Add: \alpha_2^{1,0}$	0.2	0.34	0.46
$ITE: \alpha_2^{1,1}$	0.5	0.17	0.33
$Map: \emptyset$	0	0	0

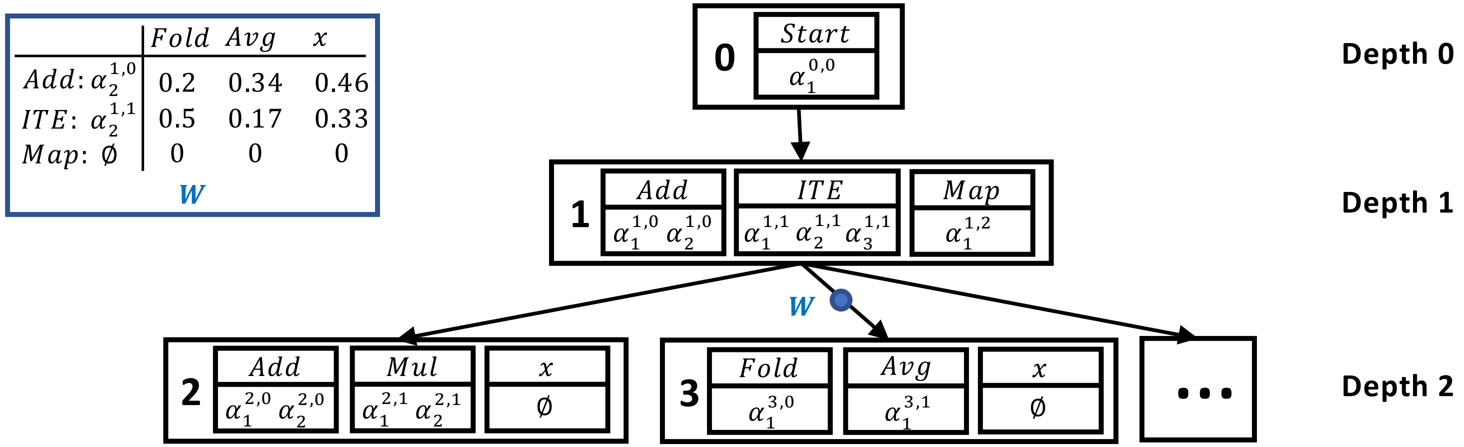
W



- Nonterminals in partial architectures of the same node ***share*** child nodes.
  - For example, the second parameters of Add $\alpha_2^{1,0}$ and ITE $\alpha_2^{1,1}$ on node 1 share child node 3.
  - Intuition – only one of the partial architectures on node 1 would be chosen in the final derivation.

Optimizing Differentiable Architecture Search

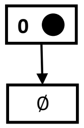
- 1. ***Node Sharing***.



- Nonterminals in partial architectures of the same node ***share*** child nodes.
  - For example, the second parameters of Add $\alpha_2^{1,0}$ and ITE $\alpha_2^{1,1}$ on node 1 share child node 3.
  - Intuition – only one of the partial architectures on node 1 would be chosen in the final derivation.
- Sharing reduces the width of a program derivation graph $T_{w,\theta}$

Optimizing Differentiable Architecture Search

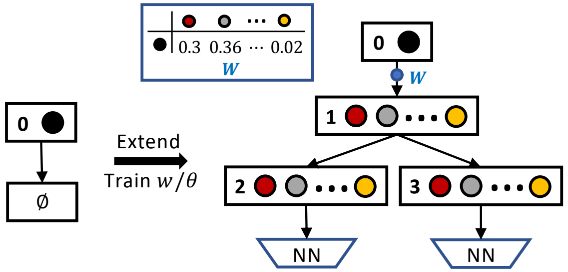
- 2. ***Iterative Graph Unfolding***.



- At each iteration, we perform two steps

Optimizing Differentiable Architecture Search

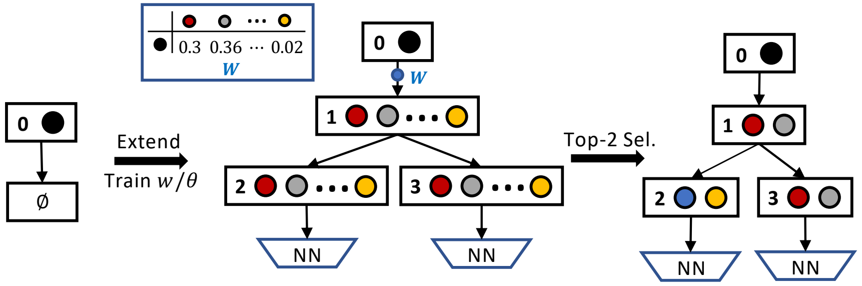
- 2. ***Iterative Graph Unfolding***.



- At each iteration, we perform two steps:
  - **Unfolding** – expand $T_{w,\theta}$ only d_s-depth deeper with any remaining nonterminals in it approximated by neural networks (d_s = 2).

Optimizing Differentiable Architecture Search

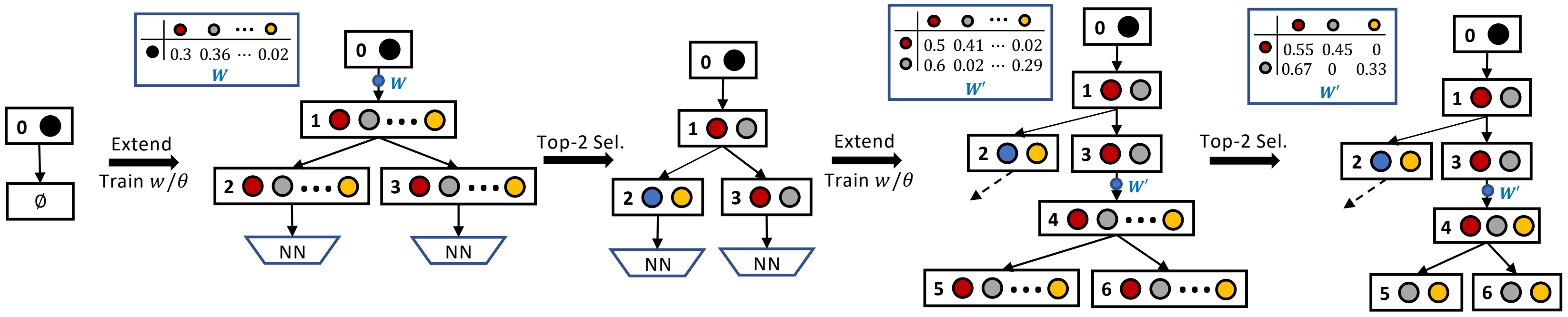
- 2. ***Iterative Graph Unfolding***.



- At each iteration, we perform two steps:
  - **Unfolding** – expand $T_{w,\theta}$ only d_s-depth deeper with any remaining nonterminals in it approximated by neural networks (d_s = 2).
  - **Top-*N* preservation** – after training an expanded $T_{w,\theta}$, on each node retain only the Top-*N* architecture derivations for each partial architecture on the node's parent (*N* = 2).
    - Top-*N* ranked by trained architecture weights

Optimizing Differentiable Architecture Search

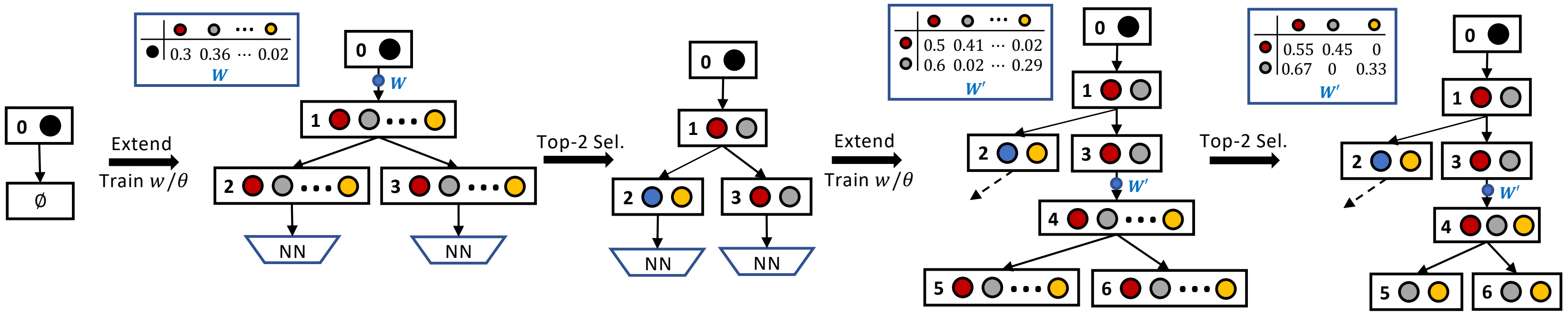
- 2. ***Iterative Graph Unfolding***.



- At each iteration, we perform two steps:
  - **Unfolding** – expand $T_{w,\theta}$ only d_s-depth deeper with any remaining nonterminals in it approximated by neural networks (d_s = 2).
  - **Top-*N* preservation** – after training an expanded $T_{w,\theta}$, on each node retain only the Top-*N* architecture derivations for each partial architecture on the node's parent (*N* = 2).
    - Top-*N* ranked by trained architecture weights

Optimizing Differentiable Architecture Search

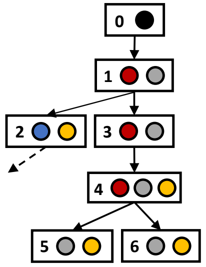
- 2. ***Iterative Graph Unfolding***.



- At each iteration, we perform two steps:
  - **Unfolding** – expand $T_{w,\theta}$ only d_s-depth deeper with any remaining nonterminals in it approximated by neural networks (d_s = 2).
  - **Top-*N* preservation** – after training an expanded $T_{w,\theta}$, on each node retain only the Top-*N* architecture derivations for each partial architecture on the node's parent (*N* = 2).
    - Top-*N* ranked by trained architecture weights
- Iterative unfolding reduces the depth of a training graph considered at each iteration.

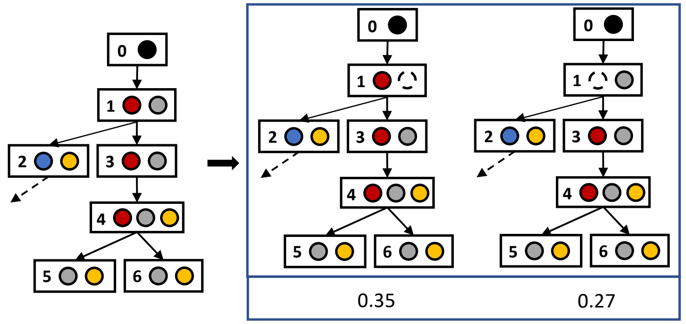
Optimizing Architecture Selection

- Upon convergence, select one discrete program from trained $T_{w,\theta}$.
  - ***Challenge*** – architecture weights may be inaccurate due to compound nodes.



Optimizing Architecture Selection

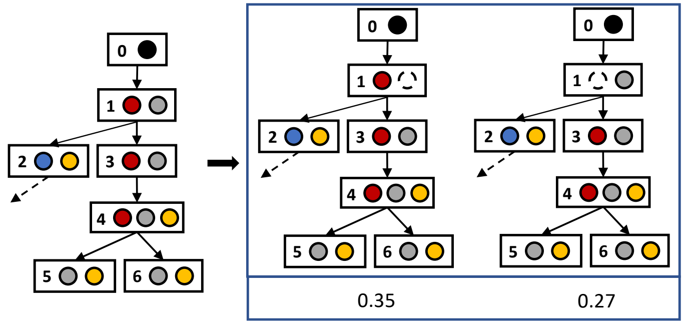
- Upon convergence, select one discrete program from trained $T_{w,\theta}$.
  - ***Challenge*** – architecture weights may be inaccurate due to compound nodes.



- Split the top-left compound node in $T_{w,\theta}$ to separate the architecture search space into disjoint partitions and train each partition until convergence.

Optimizing Architecture Selection

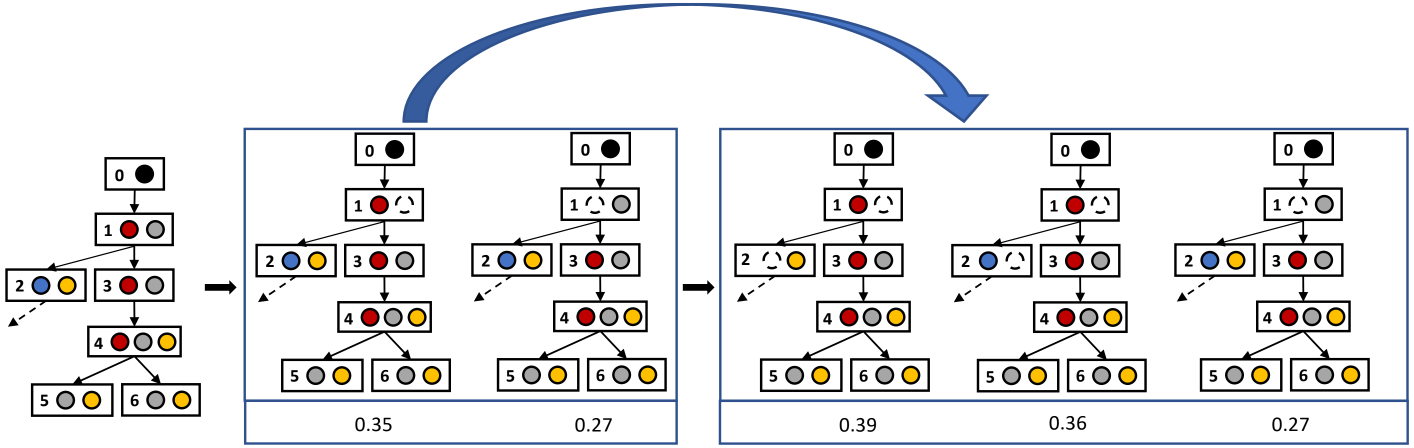
- Upon convergence, select one discrete program from trained $T_{w,\theta}$.
  - ***Challenge*** – architecture weights may be inaccurate due to compound nodes.



- Split the top-left compound node in $T_{w,\theta}$ to separate the architecture search space into disjoint partitions and train each partition until convergence.
- Maintain all partitions in a priority queue sorted by their quality (e.g., F1-score after convergence)

Optimizing Architecture Selection

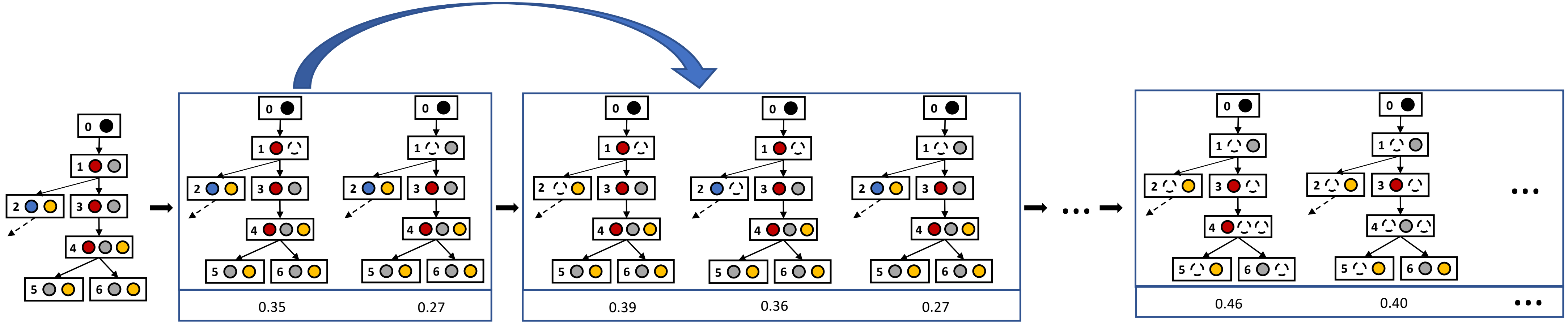
- Upon convergence, select one discrete program from trained $T_{w,\theta}$.
  - ***Challenge*** – architecture weights may be inaccurate due to compound nodes.



- Split the top-left compound node in $T_{w,\theta}$ to separate the architecture search space into disjoint partitions and train each partition until convergence.
- Maintain all partitions in a priority queue sorted by their quality (e.g., F1-score after convergence)
- Dequeue a partition from the priority queue and further split its top-left compound node.

Optimizing Architecture Selection

- Upon convergence, select one discrete program from trained $T_{w,\theta}$.
  - ***Challenge*** – architecture weights may be inaccurate due to compound nodes.



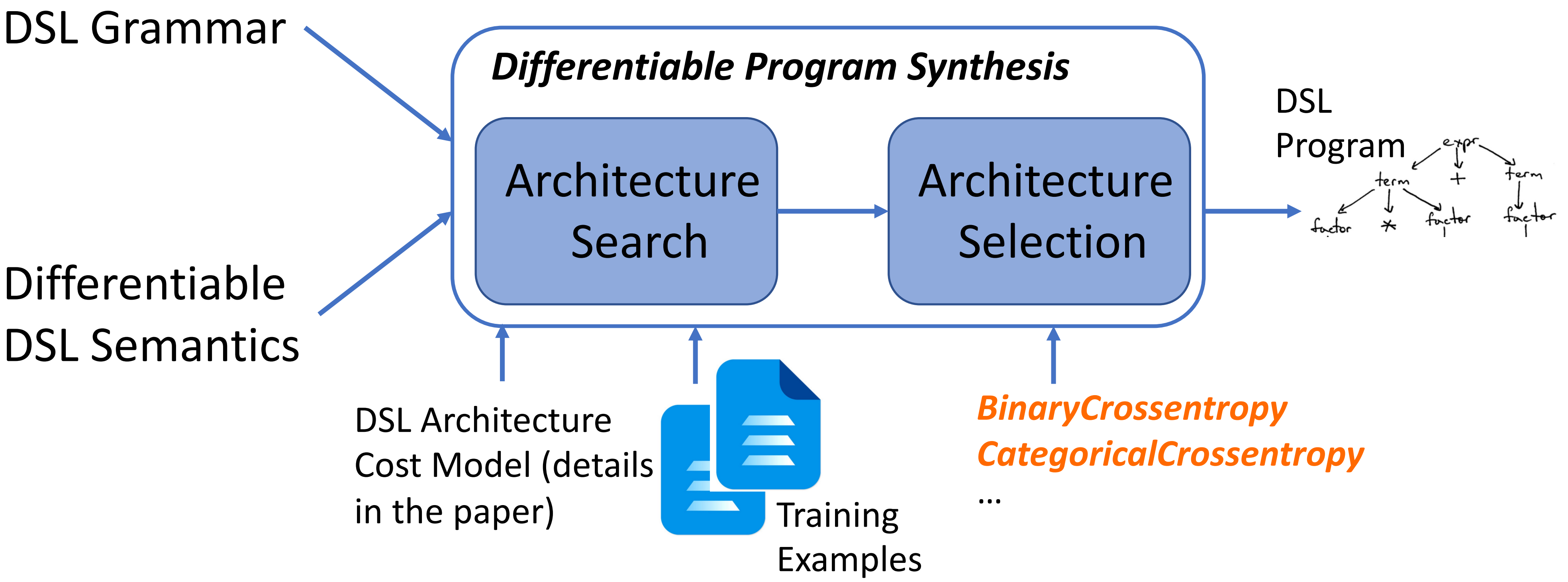
- Split the top-left compound node in $T_{w,\theta}$ to separate the architecture search space into disjoint partitions and train each partition until convergence.
- Maintain all partitions in a priority queue sorted by their quality (e.g., F1-score after convergence)
- Dequeue a partition from the priority queue and further split its top-left compound node.

Algorithm terminates when a discrete program is dequeued.

Implementation: dPads

- Implement the program learning algorithm in a tool dPads.
  dPads - **d**omain-specific **P**rogram **a**rchitecture **d**ifferentiable **s**ynthesis

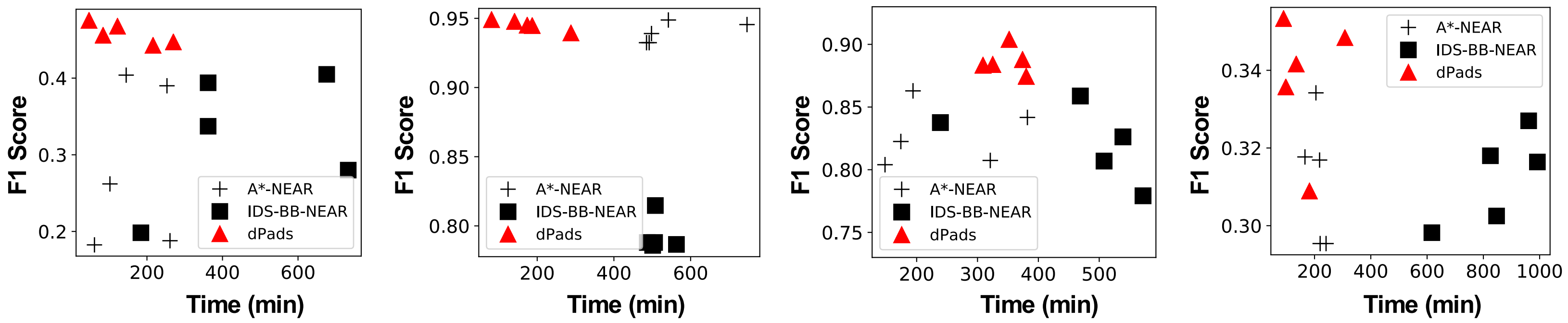
- dPads framework:
  Synthesize programs with high accuracy and low architecture cost



dPads Experiments

- Results on four sequence classification benchmarks.
  - Comparison with NEAR (a state-of-the-art program learning method based on discrete graph search)

	Crim13-sniff			Fly-vs-fly			Bball-ballhandler			Sk152-10 actions		
	F1	Acc.	Time	F1	Acc.	Time	F1	Acc.	Time	F1	Acc.	Time
RNN	.481	.851	-	.964	.964	-	.980	.980	-	.414	.428	-
A*-NEAR	.286	.820	164.92	.828	.764	**243.82**	.940	.934	553.01	.312	.315	210.23
IDS-BB-NEAR	.323	.834	463.36	.822	.750	465.57	.793	.768	513.33	.314	.317	848.44
dPads	**.458**	.812	**147.87**	**.887**	.853	348.25	**.945**	.939	**174.68**	**.337**	.337	**162.70**



(a) Crim13 (b) Basketball (c) Fly-vs-fly (d) SK152

- Differentiable program synthesis (dPads) outperforms discrete search.

Summary

- We present a novel differentiable framework for program synthesis that jointly optimizes program derivations and parameters in a continuous relaxation of the discrete program architecture search space.

- We instantiate the differentiable program synthesis framework in the context of sequence-classification tasks. Experiment results demonstrate that our program synthesizer dPads outperforms state-of-the-art program learning methods.

Thank you!