# 0_final_assignment

## jie & gao (2021): differentiable architechture search with morphism-based transformable backbone architectures


**Abstract**

This study focuses on the highest-level adaption for deep neural networks, which is at structure or architecture level. To determine the optimal neural network structures, usually we need to search for the architectures with methods like random search, evolutionary algorithm and gradient based methods. However, the existing studies mainly apply pre-defined backbones with fixed sizes, which requires certain level of manual selection and in lack of flexibility in handling incremental datasets or online data streams. This study aims at making the architecture search process more adaptive for one-shot or online training. It is extended from the existing study on differentiable neural architecture search, and we made the backbone architecture transformable rather than fixed during the training process. As is known, differentiable neural architecture search (DARTS) requires a pre-defined over-parameterized backbone architecture, while its size is to be determined manually. Also, in DARTS backbone, Hadamard product of two elements is not introduced, which exists in both LSTM and GRU cells for recurrent nets. This study introduces a growing mechanism for differentiable neural architecture search based on network morphism. It enables growing of the cell structures from small size towards large size ones with one-shot training. Two modes can be applied in integrating the growing and original pruning process. We also implement a recently proposed two-input backbone architecture for recurrent neural networks. Initial experimental results indicate that our approach and the two-input backbone structure can be quite effective compared with other baseline architectures including LSTM, in a variety of learning tasks including multi-variate time series forecasting and language modeling. On the other hand, we find that dynamic network transformation is promising in improving the efficiency of differentiable architecture search.


### darts

## 9.2 DARTS

### DIFFERENTIABLE ARCHITECTURE SEARCH (DARTS)

- Idea. We make the discretized graph choice continuous by assigning a weight alpha to each operator

  --> Continuous relaxation

- To keep alpha in (0, 1) along the path, we squish it through a softmax every time

$$x^{(j)} = \sum_{i<j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i<j} \sum_{o \in \mathcal{O}} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} exp(\alpha_{o'}^{(i,j)})} o(x^{(i)})$$

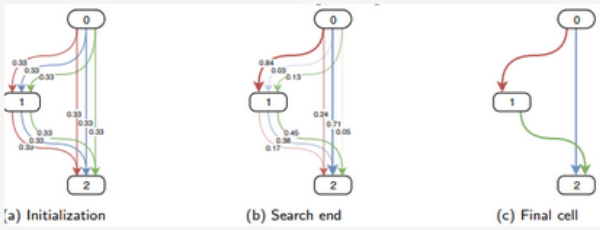
all states prior to j

kinda like importance for each operation

- As the alphas are continuous, we can use standard gradient-based methods as for usual weight learning

- In the end, we obviously want to have a single architecture

  --> Last DARTS step is to discretize again based on the learned alphas



(a) Initialization (b) Search end (c) Final cell

### DARTS ARCHITECTURE OPTIMIZATION

- In each step, DARTS solves a bi-level optimization problem

$$\begin{aligned} &\min_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \\ &s.t.\ w^*(\alpha) \in \operatorname{argmin}_w \mathcal{L}_{train}(w, \alpha) \end{aligned}$$

--> Given weights that minimize training error, find alphas that minimize validation error

- Solved via alternating SGD

no theoretical convergence guarantees yet

**Algorithm:** DARTS 1st order

**while** *not converged* **do**
  Update one-shot weights $\mathbf{w}$ by $\nabla_{\mathbf{w}} \mathcal{L}_{train}(\mathbf{w}, \alpha)$
  Update architectural parameters $\alpha$ by $\nabla_\alpha \mathcal{L}_{valid}(\mathbf{w}, \alpha)$

return $\arg\max_{o \in \mathcal{O}} \alpha_o^{(i,j)}$ for each edge $(i, j)$

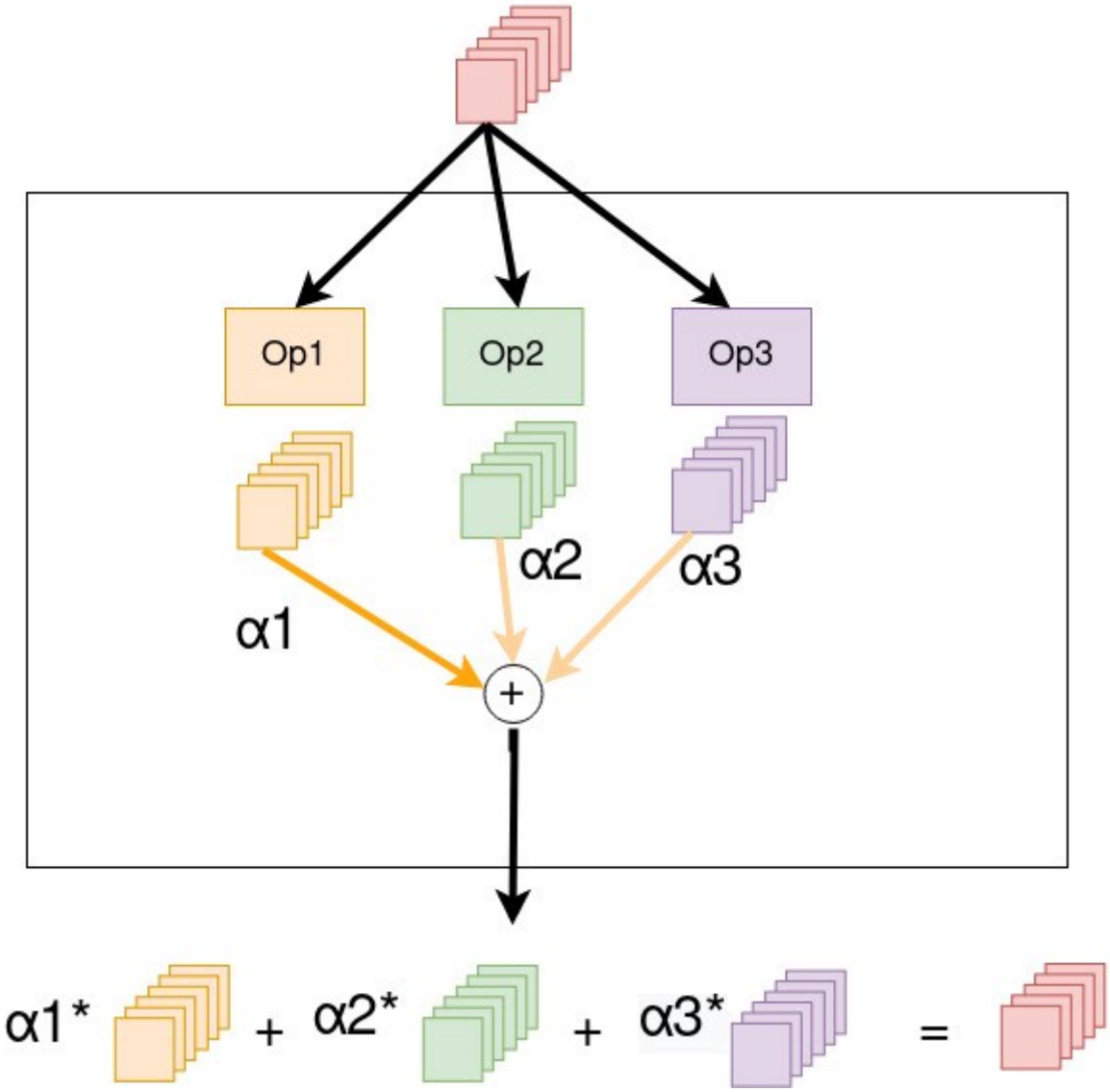
discretization

In DARTS, a cell is a Directed Acyclic Graph (DAG), it includes **k** nodes, each node connects to preceding nodes. Each edge that connects a pair of nodes **(i,j)**, applies some predefined operations (convolution, separable convolution, pooling, and more).

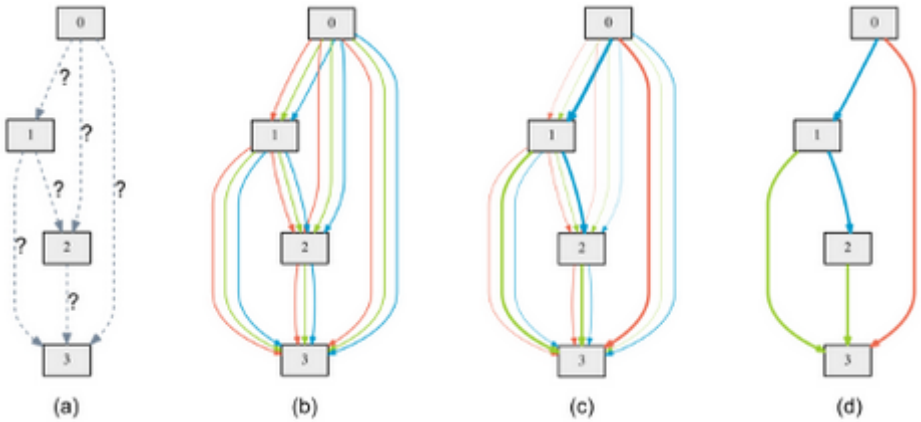
the cell search space contains **normal** and **reduction** cells, the connections between which are found by backprop

> *"To make the search space continuous, we relax the categorical choice of a particular operation to a softmax over all possible operations:" — DARTS*

output is **mixed operation**




in the end, we slap an **argmax** over these and just retain the one most likely to improve performance




[blogpost]

[blogpost]

the **bi-level training procedure** is approximated (would otherwise be extremely costly to conduct the entire training each time $\alpha$ is updated)

> *"The idea is to approximate $w^*(\alpha)$ by adapting $w$ using only a single training step, without solving the inner optimization completely by training until convergence."*

---

### identified shortcomings

---

authors address

1. standard darts requires fixing the size of the backbone architecture a priori, which is hard and contradicts the automl idea
2. the over-parameterized architecture can then only be pruned but not grown, precluding improvement if the architecture was chosen too small
3. each operation can only be applied once per node

---

### main contributions

---

In this study, we propose a hybrid method of automatically growing the architecture during the learning process of gradient-based one-shot architecture search. We introduce the technique of network morphism to keep the learned mapping as much as possible in each growing transformation. The main contributions include:

1. dynamic growing & pruning mechanisms
2. network morphisms for both node growing & dynamic operation transformation
3. two-input backbone architecture for rnn (i.e., new cell space?)
4. pruning mechanisms for the process of cell structures growing

---

### network morphisms

---

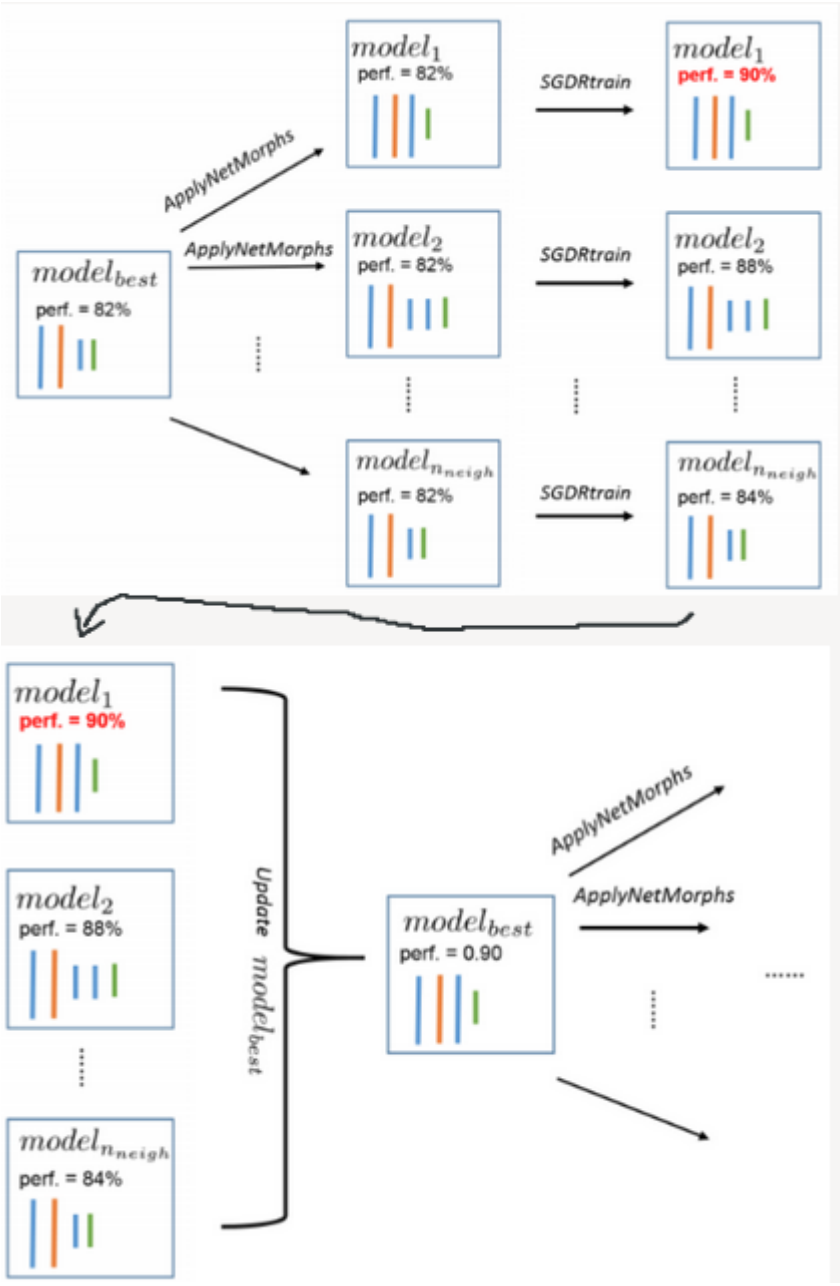
- Idea. Take a network and apply local changes that retain output

  Examples: “Net2DeeperNet”, “Net2WiderNet”, etc.

- Then, perform a few training iterations for each of the morphed nets

  --> Keep the best one & start over

  --> Relatively cheap bc of weight inheritance from previous net




## new search space

proposed in predecessor paper

## A. A Novel Search Space for Recurrent Cells

First, we introduce a novel backbone architecture with two inputs and one output that is motivated by the cell structures used in RNN models. The diagram of the proposed backbone architecture is shown in Fig. 1. Different from the backbone structure of the directed acyclic graph (DAG) implemented in DARTS, we introduce a new backbone structure in which each node has two inputs, and each operation in a node takes both two inputs as an interaction function. The corresponding formula for the information flow in each cell is given by

$$
\begin{aligned}
\bar{o}_1(x_t, h_{t-1}) &= \sum_{o_1 \in O} w_1^{o_1} o_1(x_t, h_{t-1}) \\
\bar{o}_j(x_t, h_{t-1}) &= \sum_{o_j \in O} w_j^{o_j} o_j(x_t, \bar{o}_{j-1}(x_t, h_{t-1})), \\
w_j^o &= \frac{\exp(\alpha_j^{o_j})}{\sum_{o_j \in O} \exp(\alpha_j^{o_j})} \\
h_t &= \bar{o}_J(x_t, h_{t-1}) \\
j &= 2, \ldots, J
\end{aligned}
\tag{3}
$$

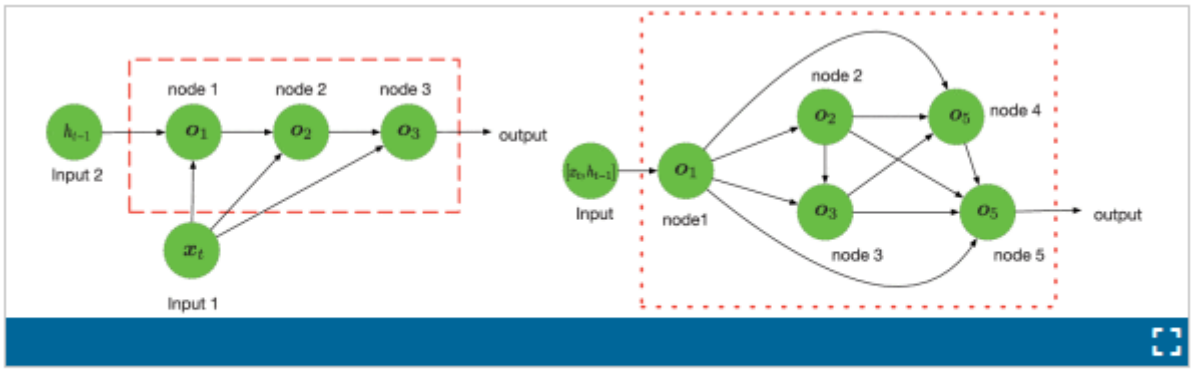



**FIGURE 1.**
Diagram of the newly proposed backbone architecture (left) and DAG backbone for DARTS (right).

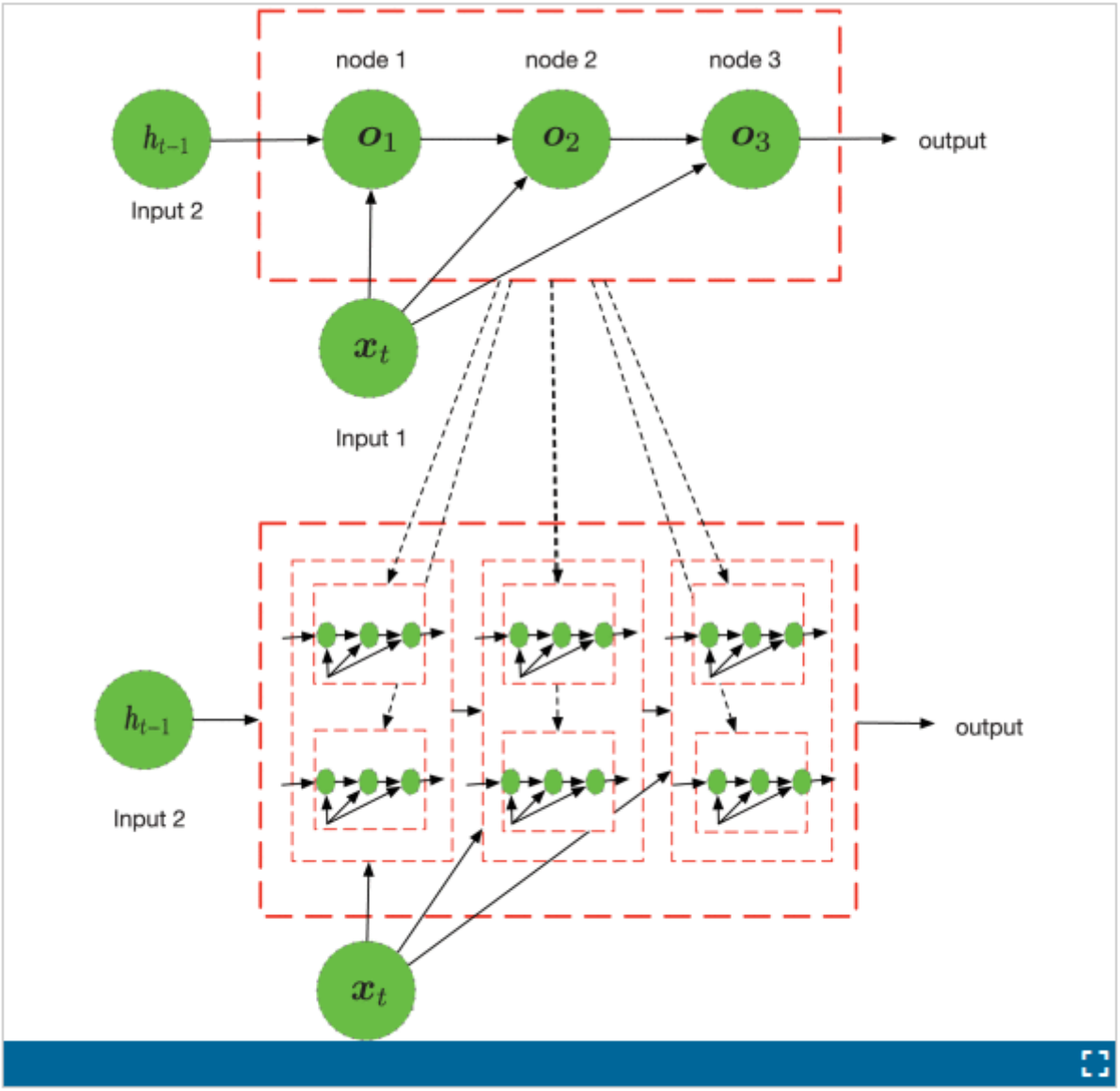



**FIGURE 2.**
Hierarchical structure of the proposed RNN backbone cells.

code for available on [github]

## Requirements

```
Python >= 3.5.5, PyTorch == 0.3.1, torchvision == 0.2.0
```

NOTE: PyTorch 0.4 is not supported at this moment and would lead to OOM.

## Datasets

Instructions for acquiring PTB and WT2 can be found here. While CIFAR-10 can be automatically downloaded by torchvision, ImageNet needs to be manually downloaded (preferably to a SSD) following the instructions here.

## Pretrained models

The easist way to get started is to evaluate our pretrained DARTS models.

**CIFAR-10** (cifar10_model.pt)

```
cd cnn && python test.py --auxiliary --model_path cifar10_model.pt
```

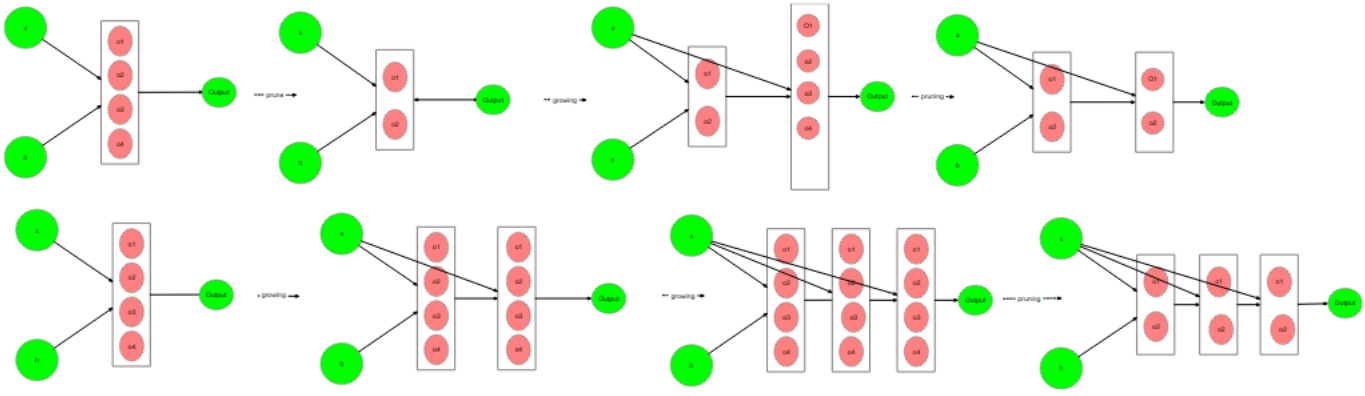
also microsoft nni has stuff here (e.g., fixed darts cells, also entire algorithms apparently, pruning, morphisms, ...)

---

### strategies

---

1. prune at each stage
2. prune at the end



**Algorithm 2** The Algorithm of Growing Architecture Search

**Require:** Initial architecture $A_0$, $\theta_0$

Set eva_best = a large number
$A = A_0, \theta = \theta_0$
**for** $i$ in $1, 2, 3, ..., S$ **do**
  Backbone training: $A, \theta = \text{train}(A, D_t, D_v)$
  **if** Pruning at each stage **then**
    Pruning: $A = \text{Prune}(A, \theta)$
    Fine tuning: $\theta = \text{Tune}(A, D_t, \theta)$
  **end if**
  Evaluate: eva = $\text{Evaluate}(A, \theta, D_{test})$
  **if** eva < eva_best **then**
    $A_{\text{best}} = A$
    $A, \theta = \text{update}(A, \theta)$
    eva_best = eva
  **else**
    break
  **end if**
**end for**
**if** Prune at the end **then**
  Pruning: $A = \text{Prune}(A_{\text{best}}, \theta)$
  Evaluate: eva = $\text{Evaluate}(A, D_{test})$
**end if**

**Algorithm 3** The Algorithm of update function of nodes

**Require:** Previous architecture $A_{\text{prev}}$.
**Require:** Previous model parameters $\theta_{\text{prev}}$.

Initialize expanded architecture $A_{\text{expanded}}$.
Inherit the parameter from the previous model for existing nodes.
Initialize the newly added parameters in the expanded model with network morphism discussed in Section 3.2.1.
**return** $A_{\text{expanded}}$, $\theta_{\text{expanded}}$

**Algorithm 4** The Algorithm of update function of operations

**Require:** Previous architecture $A_{\text{prev}}$
**Require:** Previous model parameters $p_{\text{prev}}$

1: **if** Threshold of evolving operations reached during the training process **then**
2: Select the operation to be duplicated by checking absolute combination weights or the number/summation of negative minimum eigen values for splitting matrices.
3: Initialize expanded operations.
4: Inherit the parameters from the previous model.
5: Initialize the combination weights of the newly added operation, as well as newly added model parameters in the expanded model with network morphism or steepest descent splitting discussed in Section 3.2.2.
6: **end if**
7: **return** $A_{\text{expanded}}, \theta_{\text{expanded}}$

---

**to do**

---

- build new two-to-one cell
- enable network morphisms
  - growing of nodes. when adding a new node, we need to ensure the similarity of the output and the previous output (via specific initialization + additive noise to avoid true-zero weights) - slightly different for standard darts and new two-to-one backbone $\rightarrow$ perhaps only one option
  - growing of operators. using different number of nodes and applying different numbers of operations in different nodes $\rightarrow$ potentially combination of two or more of the same operation (apparently beneficial)

    where we take a single-node cell for example with $o_j = o_0 = h_{t-1}$. Since the weighted combination of these two operations can not be fully expressed by one operation with certain weights in any circumstances, this will actually increase the model expressiveness. It can be considered as making an ensemble of layers with shared output activation.
- enable dynamic pruning

  pruning process based on the combination weights of operations, which means when the weight for a candidate operation is less than a certain threshold, the corresponding operation will be removed and weights will be reallocated for other operations. With dynamic growing and pruning mechanisms of operations, architecture transformation can be more adaptive to the training process as well as the learning tasks by making use of weights information.
- include replacing & resampling - don't get what they mean

- implement criteria for growth
  - Criterion for growing of nodes: We implement two-level early stopping criteria to determine when a new node will be added in the architecture, and when the whole growing process should be stopped. That is when the validation performance during the training of backbone architecture is no longer increasing for several epochs, we add a new node to the backbone. When adding new node no longer increases the model performance on validation set, when stop the whole growing process and select the optimal architecture. Similar criterion for architecture updating is also implemented in (Wu et al., 2019).
  - Criterion for operator growing: We add an other operation that has the largest weights when its weights surpass a certain threshold under the strategy of network morphism. Under steepest descent strategy, when the validation loss is no-longer decreasing, we can select the operation with the largest number of negative minimum eigen values of splitting matrices in their parameter vectors. If all the hidden states have negative minimum eigen values for multiple operations in each path, we can split these operations simultaneously or split the one with minimum summation of minimum eigen values. These can be considered as different settings.
  - Criterion for operator pruning: We prune an operation in a node when its corresponding weight is close to 0, while the weight of the operation with the largest weight does not surpass a threshold.
- implement pruning at each stage (darts does it at the end)

they experiment with

- multivariate ts forecasting (rnn) - the datasets being studied are the stock indices returns of G7 countries andBRICS countries in the last ten years, while the task is one-step-ahead prediction for thereturn vectors of all indices
- language modeling (rnn)
- image classification (cnn) - results are not documented though

→ perhaps exclude operator growth (authors also do pure node growing)