NETWORK ARCHITECTURE SEARCH & COMPRESSION

Rundong Li

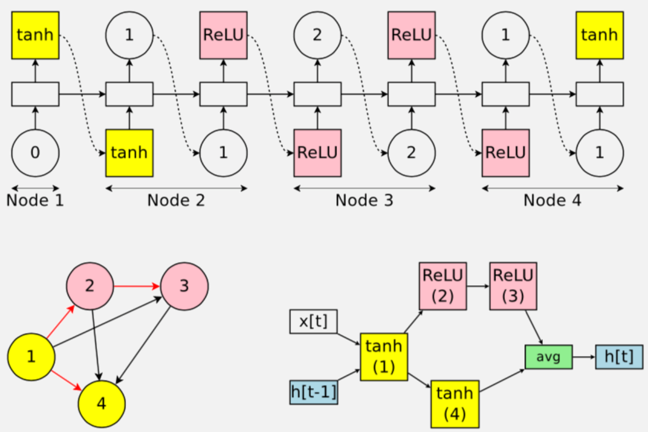
ShanghaiTech University

PAPERS TODAY

- Differentiable methods:
  - DARTS: Differentiable Architecture Search [arXiv]
  - Differentiable Fine-grained Quantization [arXiv]
  - You Only Search Once: Single Shot Neural Architecture Search via Direct Sparse Optimization [arXiv] [OpenReview]
  - Neural Architecture Optimization [arXiv]
- RL based methods:
  - HAQ: Hardware-Aware Automated Quantization [arXiv]

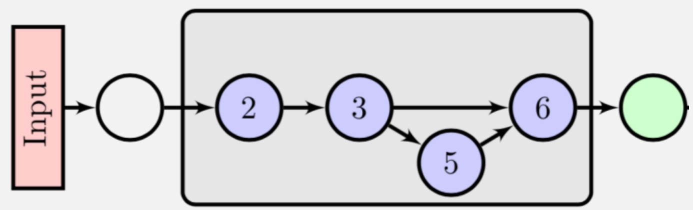
ARCHITECTURE SEARCH (NAS)

REINFORCEMENT LEARNING



Given computation graph node, Agent *(top, usually LSTM)* predicts the preceding node(s) and operation type. Sampled ops and edges form a building cell (bottom).

EVOLUTIONARY APPROACH

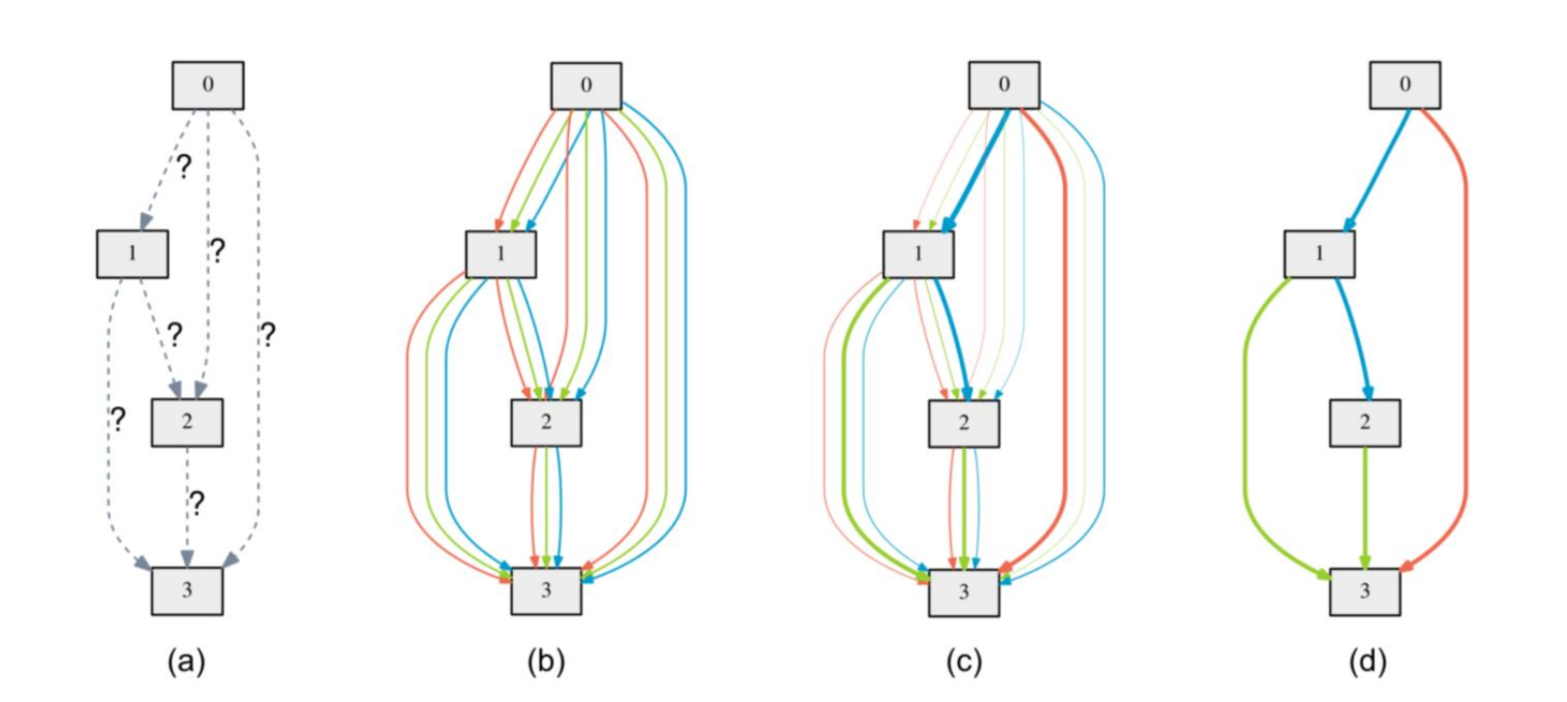


$$\boldsymbol{x}_o^{(1)} = 0\text{-}01\text{-}000\text{-}0010\text{-}00101\text{-}0$$

Encoding building blocks into *genotype,* then apply evolution operations, finally decode back to *phenotypes* (building block).

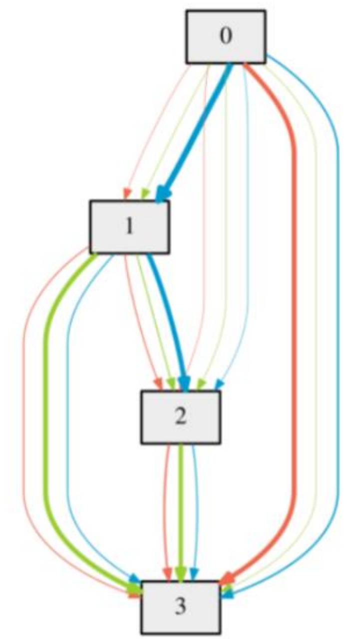
教练，我想用 SGD…

DIFFERENTIABLE ARCHITECTURE SEARCH (DARTS)



Rectangles: intermediate feature maps;
Arrows: operations (e.g. Conv3x3, DepthWiseConv3x3, Identity…);
Thickness: architecture parameters.

DARTS: ARCHITECTURE



(c)

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

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

- α : architecture parameter
- w : model parameter (weights)
- $w^*(\alpha)$: optimal weights on given architecture

DARTS: OPTIMIZATION

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i,j)
while *not converged* **do**
  1. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
  2. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}}\, \alpha_o^{(i,j)}$ for each edge (i,j)

- It's prohibitive to optimize: $w^*(\alpha)$ varies whenever α updated;
- Approximation: relax $w^*(\alpha)$ into one-step update of w on training set: $w' = w - \varepsilon \nabla_w \mathcal{L}_{train}(w, \alpha)$;
- Problem: when taking derivative w.r.t. α: first derivative $\frac{\partial L}{\partial w'}$, then $\frac{\partial w'}{\partial \alpha}$…
- WTF, it's **Hessian**…

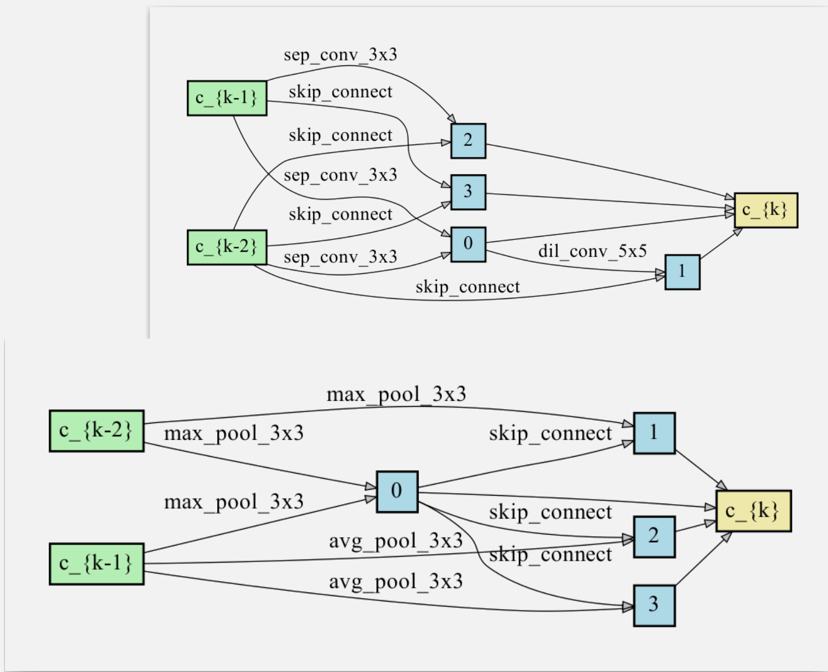
DARTS: OPTIMIZATION

, $w^+ = w + \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$ and $w^- = w - \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$. Then:

$$\nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon}$$

- It's prohibitive to optimize: $w^*(\alpha)$ varies whenever α updated;
- Approximation: relax $w^*(\alpha)$ into one-step update of w on training set: $w' = w - \varepsilon \nabla_w \mathcal{L}_{train}(w, \alpha)$;
- Problem: when taking derivative w.r.t. α: first derivative $\frac{\partial L}{\partial w'}$, then $\frac{\partial w'}{\partial \alpha}$…
- WTF, it's **Hessian**…

DARTS: RESULTS



On CIFAR10: the normal cell (top) and the reduction cell (bottom, with stride 2).

On CIFAR10	Top-1 Acc
Paper (2nd order)	97.17 $\pm$ 0.06
Paper (1st order)	97.06
Code README	97.24 $\pm$ 0.9
Official code	97.14

DARTS: PROS AND CONS

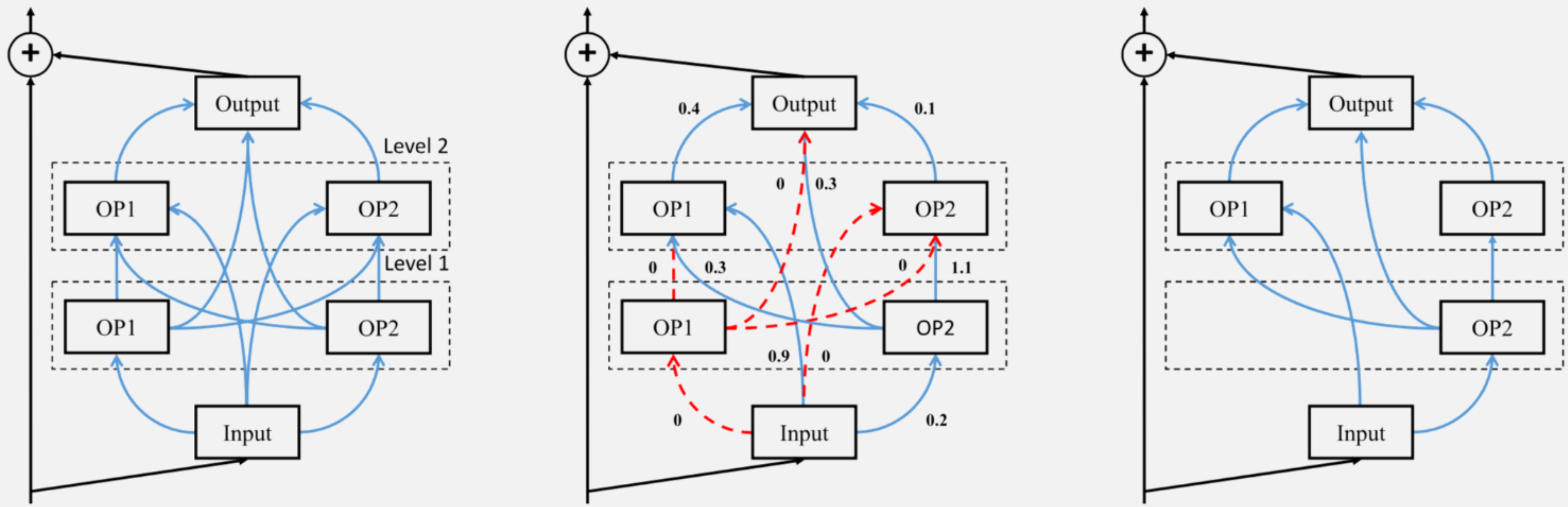
PROS.

- Fast to converge: take ~18 hours on single 1080Ti (RL methods usually take *thousands* GPU hours);
- Intuitive architecture and learning process;

CONS.

- Does the 2nd derivative really necessary? (*no 1st vs. 2nd reports on ImageNet*)
- Only able to search *small building blocks* (takes ~11G GRAM when searching CIFAR10) ;
- Not able to search blocks with different *channel numbers*;

YOU ONLY SEARCH ONCE



(a) (b) (c)

a) NAS as *pruning*: prune the completely connected block;
b) In the search process, jointly optimize the weights and the scale λ associated with each edge. Note that λ is *sparse regularized (L1)* during training;
c) The final model after removing useless connections and operations;

DIFFERENTIABLE FINE-GRAINED QUANTIZATION

$$q_i = \frac{\sum_{j=0}^{1} exp(\alpha_{i_j})\mathcal{B}(q_{i_j})}{\sum_{j=0}^{1} exp(\alpha_{i_j})},$$

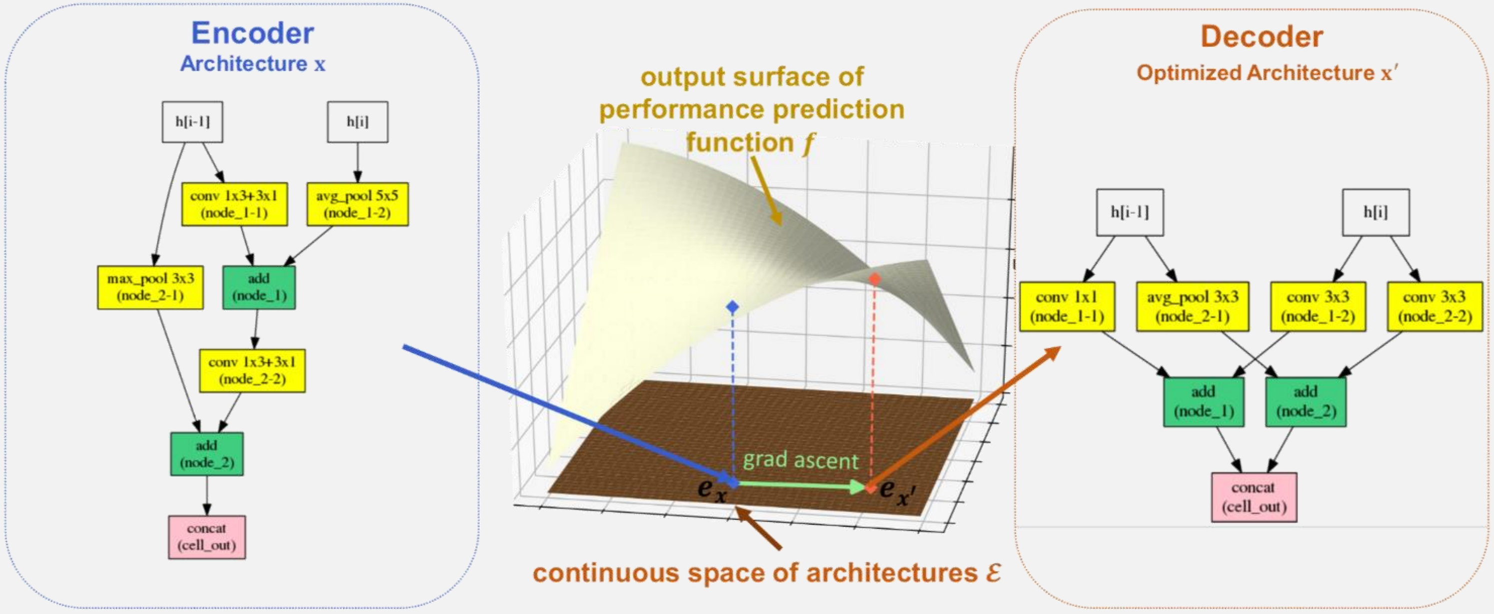
$$\min_{\alpha} \mathcal{G}(\alpha),$$

$$s.t. \quad \mathcal{L}_{val}(w^*, \alpha) - \theta \leq 0,$$

$$w^* = \arg\min_{w} \mathcal{L}_{train}(w^*, \alpha).$$

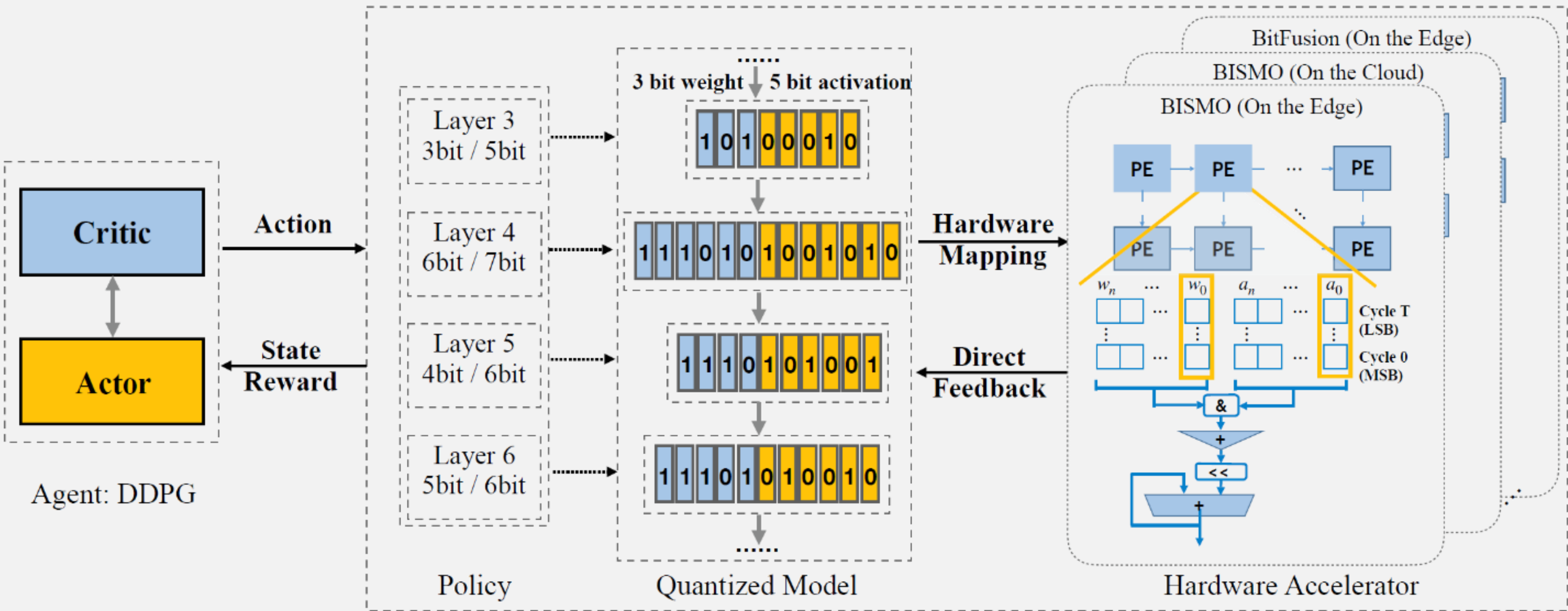
- Almost identical structure to DARTS;
- Focus on minimize model size $\mathcal{G}(\alpha)$;
- Candidate bit-width: binary and 8-bits on VGG-16;

NEURAL ARCHITECTURE OPTIMIZATION



- Encoder (E, vanilla LSTM): *string* description of architecture $\mathcal{X}$ into continuous embedding ε;
- Predictor (P, regression FC net): predict performance s from embedding ε;
- Take **SGD** on P's gradient to get optimal ε^*!
- Decoder (D, attention LSTM): mapping ε^* back to architecture $\mathcal{X}^*$;

HAQ: HARDWARE AWARE AUTOMATED QUANTIZATION



HAQ: HARDWARE AWARE AUTOMATED QUANTIZATION

- Observations (Conv and FC layers):

$$O_k = (k, c_{\text{in}}, c_{\text{out}}, s_{\text{kernel}}, s_{\text{stride}}, s_{\text{feat}}, n_{\text{params}}, i_{\text{dw}}, i_{\text{w/a}}, a_{k-1})$$
$$O_k = (k, h_{\text{in}}, h_{\text{out}}, 1, 0, s_{\text{feat}}, n_{\text{params}}, 0, i_{\text{w/a}}, a_{k-1})$$

- Actions: *continues* bit-width factor a_k,

$$b_k = \text{round}(b_{\text{min}} - 0.5 + a_k \times (b_{\text{max}} - b_{\text{min}} + 1))$$

- Rewards: *accuracy drop* and real hardware feedback,
  - ``If the current policy exceeds our resource budget (on latency, energy or model size), we will sequentially decrease the bitwidth of each layer until the constraint is finally satisfied.''