Searching for Calibrated, Efficient, and Expressive Neural Networks

Arber Zela

University of Freiburg & ELLIS
zelaa@cs.uni-freiburg.de

Nov 2024

A Few Words About Me

- Apr 1993: Opened my eyes for the first time in Tirana, Albania.

A Few Words About Me

- Apr 1993: Opened my eyes for the first time in Tirana, Albania.
- Apr 2015: Moved to Freiburg, Germany for my Master in CS.

A Few Words About Me

- Apr 1993: Opened my eyes for the first time in Tirana, Albania.
- Apr 2015: Moved to Freiburg, Germany for my Master in CS.
- Feb 2019: Started my PhD at the University of Freiburg under the supervision of Frank Hutter.

A Few Words About Me

- Apr 1993: Opened my eyes for the first time in Tirana, Albania.
- Apr 2015: Moved to Freiburg, Germany for my Master in CS.
- Feb 2019: Started my PhD at the University of Freiburg under the supervision of Frank Hutter.
- I am also an ELLIS PhD student co-supervised by Yee Whye Teh (University of Oxford).
  - Spent also 1 year in UK (including industry internship).

A Few Words About Me

- Apr 1993: Opened my eyes for the first time in Tirana, Albania.
- Apr 2015: Moved to Freiburg, Germany for my Master in CS.
- Feb 2019: Started my PhD at the University of Freiburg under the supervision of Frank Hutter.
- I am also an ELLIS PhD student co-supervised by Yee Whye Teh (University of Oxford).
  - Spent also 1 year in UK (including industry internship).
- **During my PhD:**
  - I worked a lot on improving algorithms that search for architectures of deep neural networks.
  - Mostly empirical and applied to various settings.

A Few Words About Me

- Apr 1993: Opened my eyes for the first time in Tirana, Albania.
- Apr 2015: Moved to Freiburg, Germany for my Master in CS.
- Feb 2019: Started my PhD at the University of Freiburg under the supervision of Frank Hutter.
- I am also an ELLIS PhD student co-supervised by Yee Whye Teh (University of Oxford).
  - Spent also 1 year in UK (including industry internship).
- **During my PhD:**
  - I worked a lot on improving algorithms that search for architectures of deep neural networks.
  - Mostly empirical and applied to various settings.
  - Also adopted a cat.

Motivation

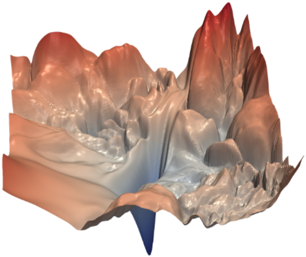
Why does the architecture matter?

- **Expressive power:** What the model can learn

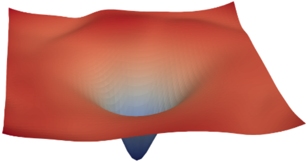
Motivation

Why does the architecture matter?

- **Expressive power:** What the model can learn
- **Optimization:** How fast it can learn
  - E.g., skip connections for vanishing gradients and smoother loss landscape.



(a) without skip connections



(b) with skip connections

Li et al. Visualizing the Loss Landscape of Neural Nets. In NeurIPS 2018

Motivation

Why does the architecture matter?

- **Expressive power:** What the model can learn
- **Optimization:** How fast it can learn
  - E.g., skip connections for vanishing gradients and smoother loss landscape.
- **Generalization:** How well it performs on unseen data
  - Prior: Search space; architectural preferences (e.g., conv layers for image data)
  - Posterior: Updated belief given training data

Motivation

Why does the architecture matter?

- **Expressive power:** What the model can learn
- **Optimization:** How fast it can learn
  - E.g., skip connections for vanishing gradients and smoother loss landscape.
- **Generalization:** How well it performs on unseen data
  - Prior: Search space; architectural preferences (e.g., conv layers for image data)
  - Posterior: Updated belief given training data
- **Practical implications**

Motivation

Why does the architecture matter?

- **Expressive power:** What the model can learn
- **Optimization:** How fast it can learn
  - E.g., skip connections for vanishing gradients and smoother loss landscape.
- **Generalization:** How well it performs on unseen data
  - Prior: Search space; architectural preferences (e.g., conv layers for image data)
  - Posterior: Updated belief given training data
- **Practical implications**
  - **Efficiency**: memory usage, inference speed, etc.

Motivation

Why does the architecture matter?

- **Expressive power:** What the model can learn
- **Optimization:** How fast it can learn
  - E.g., skip connections for vanishing gradients and smoother loss landscape.
- **Generalization:** How well it performs on unseen data
  - Prior: Search space; architectural preferences (e.g., conv layers for image data)
  - Posterior: Updated belief given training data
- **Practical implications**
  - **Efficiency**: memory usage, inference speed, etc.
  - **Robustness**: to noise, distribution shift, etc.

Motivation

Why does the architecture matter?

- **Expressive power:** What the model can learn
- **Optimization:** How fast it can learn
  - E.g., skip connections for vanishing gradients and smoother loss landscape.
- **Generalization:** How well it performs on unseen data
  - Prior: Search space; architectural preferences (e.g., conv layers for image data)
  - Posterior: Updated belief given training data
- **Practical implications**
  - **Efficiency**: memory usage, inference speed, etc.
  - **Robustness**: to noise, distribution shift, etc.
  - **Interpretability**: simpler models easier to analyze.

How to search for architectures in a more systematic way?

Agenda

1. Problem Formulation
2. Optimization and Generalization: Differentiable Architecture Search
3. Robustness: Neural Ensemble Search
4. Efficiency: Multi-objective Differentiable Architecture Search
5. Expressivity: Linear RNNs for State-tracking

Problem Formulation

Bi-level optimization problem

Assume we have a discrete architecture space $\mathcal{A}$, a loss function $\mathcal{L}$, and a dataset $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{valid}$.

- $\mathcal{L}$ is a (stochastic) function of both $\lambda \in \mathcal{A}$ and $\mathbf{w} \in \mathbb{R}^d$
- Optimizing both $\mathcal{L}_T := \mathcal{L}(\cdot; \mathcal{D}_{train})$ and $\mathcal{L}_V := \mathcal{L}(\cdot; \mathcal{D}_{valid})$ corresponds to a bi-level optimization problem:

$$\min_{\lambda} \{\mathcal{L}_V^*(\lambda) := \mathcal{L}_V(\mathbf{w}^*(\lambda), \lambda)\} \qquad \text{(upper-level)}$$

$$s.t. \quad \mathbf{w}^*(\lambda) = \arg\min_{\mathbf{w}} \mathcal{L}_{train}(\mathbf{w}, \lambda), \qquad \text{(lower-level)}$$

Problem Formulation

Bi-level optimization problem

Assume we have a discrete architecture space $\mathcal{A}$, a loss function $\mathcal{L}$, and a dataset $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{valid}$.

- $\mathcal{L}$ is a (stochastic) function of both $\lambda \in \mathcal{A}$ and $\mathbf{w} \in \mathbb{R}^d$
- Optimizing both $\mathcal{L}_T := \mathcal{L}(\cdot; \mathcal{D}_{train})$ and $\mathcal{L}_V := \mathcal{L}(\cdot; \mathcal{D}_{valid})$ corresponds to a bi-level optimization problem:

$$\min_{\lambda} \{\mathcal{L}_V^*(\lambda) := \mathcal{L}_V(\mathbf{w}^*(\lambda), \lambda)\} \qquad \text{(upper-level)}$$

$$s.t. \quad \mathbf{w}^*(\lambda) = \arg\min_{\mathbf{w}} \mathcal{L}_{train}(\mathbf{w}, \lambda), \qquad \text{(lower-level)}$$

Solvers for the upper-level:

- 0-th order methods: e.g., evolutionary strategies, Bayesian optimization.
- 1-th order methods: SGD on continuous relaxation of $\mathcal{A}$ using hypergradients.

Problem Formulation
Bi-level optimization problem

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \left(\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}\right)\Bigg|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} =$$

$$\underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}}_{\text{hyperparam direct grad.}} + \overbrace{\underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})}}_{\text{parameter direct grad.}} \times \underbrace{\frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{best-response Jacobian}}}^{\text{hyperparam indirect grad.}} \tag{3}$$

Lorraine et al. Optimizing Millions of Hyperparameters by Implicit Differentiation. In AISTATS 2020

Problem Formulation

Bi-level optimization problem

$$\underbrace{\frac{\partial \mathcal{L}_{\mathrm{V}}^{*}(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \left(\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^{*}}{\partial \boldsymbol{\lambda}} \right) \Bigg|_{\boldsymbol{\lambda}, \mathbf{w}^{*}(\boldsymbol{\lambda})} =$$

$$\underbrace{\frac{\partial \mathcal{L}_{\mathrm{V}}(\boldsymbol{\lambda}, \mathbf{w}^{*}(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}}_{\text{hyperparam direct grad.}} + \overbrace{\underbrace{\frac{\partial \mathcal{L}_{\mathrm{V}}(\boldsymbol{\lambda}, \mathbf{w}^{*}(\boldsymbol{\lambda}))}{\partial \mathbf{w}^{*}(\boldsymbol{\lambda})}}_{\text{parameter direct grad.}} \times \underbrace{\frac{\partial \mathbf{w}^{*}(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{best-response Jacobian}}}^{\text{hyperparam indirect grad.}} \tag{3}$$

Theorem 1 (Cauchy, Implicit Function Theorem). *If for some* $(\boldsymbol{\lambda}', \mathbf{w}')$, $\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'} = 0$ *and regularity conditions are satisfied, then surrounding* $(\boldsymbol{\lambda}', \mathbf{w}')$ *there is a function* $\mathbf{w}^{*}(\boldsymbol{\lambda})$ *s.t.* $\frac{\partial \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}, \mathbf{w}^{*}(\boldsymbol{\lambda})} = 0$ *and we have:*

$$\frac{\partial \mathbf{w}^{*}}{\partial \boldsymbol{\lambda}}\Bigg|_{\boldsymbol{\lambda}'} = -\underbrace{\left[\frac{\partial^{2} \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \mathbf{w}^{T}} \right]^{-1}}_{\text{training Hessian}} \times \underbrace{\frac{\partial^{2} \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w} \partial \boldsymbol{\lambda}^{T}}}_{\text{training mixed partials}} \Bigg|_{\boldsymbol{\lambda}', \mathbf{w}^{*}(\boldsymbol{\lambda}')} \quad \text{(IFT)}$$

Lorraine et al. Optimizing Millions of Hyperparameters by Implicit Differentiation. In AISTATS 2020

Problem Formulation

Bi-level optimization problem

$$\frac{\partial \mathcal{L}_{\mathrm{V}}^{*}}{\partial \boldsymbol{\lambda}} = \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^{*}}{\partial \boldsymbol{\lambda}}$$

$$= \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} + \underbrace{\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \; -\left[\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}\,\partial \mathbf{w}^T}\right]^{-1}}_{\text{vector-inverse Hessian product}} \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}\,\partial \boldsymbol{\lambda}^T}$$

$$= \frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \boldsymbol{\lambda}} + \underbrace{\frac{\partial \mathcal{L}_{\mathrm{V}}}{\partial \mathbf{w}} \times -\left[\frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}\,\partial \mathbf{w}^T}\right]^{-1} \frac{\partial^2 \mathcal{L}_{\mathrm{T}}}{\partial \mathbf{w}\,\partial \boldsymbol{\lambda}^T}}_{\text{vector-Jacobian product}}$$

Lorraine et al. Optimizing Millions of Hyperparameters by Implicit Differentiation. In AISTATS 2020

Problem Formulation

Bi-level optimization problem

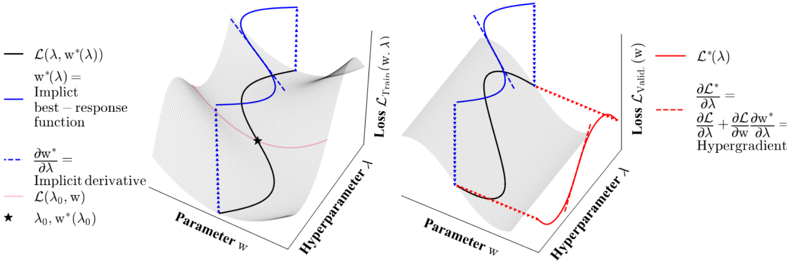
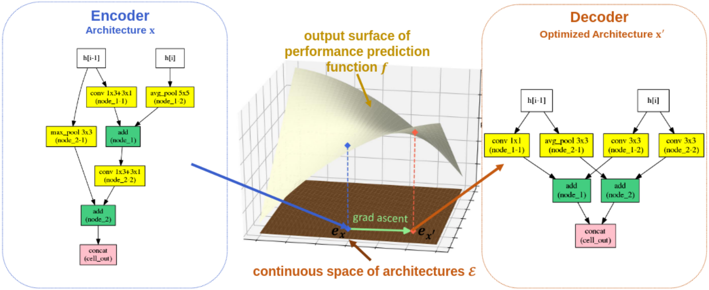


Figure 1: Overview of gradient-based hyperparameter optimization (HO). *Left:* a training loss manifold; *Right:* a validation loss manifold. The implicit function $\mathbf{w}^*(\boldsymbol{\lambda})$ is the best-response of the weights to the hyperparameters and shown in blue projected onto the $(\boldsymbol{\lambda}, \mathbf{w})$-plane. We get our desired objective function $\mathcal{L}_V^*(\boldsymbol{\lambda})$ when the best-response is put into the validation loss, shown projected on the hyperparameter axis in red. The validation loss does not depend directly on the hyperparameters, as is typical in hyperparameter optimization. Instead, the hyperparameters only affect the validation loss by changing the weights' response. We show the best-response Jacobian in blue, and the hypergradient in red.

Lorraine et al. Optimizing Millions of Hyperparameters by Implicit Differentiation. In AISTATS 2020

Differentiable Architecture Search

- Between 2 nodes (features maps): Categorical choice for which operation to use.
  - Relax discrete space using a convex combination of these choices $\longrightarrow$ supernetwork with shared weights between architectures

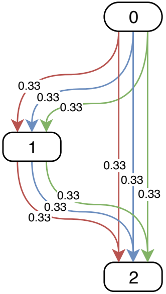


Liu et al. Neural Architecture Optimization. In NeurIPS 2018

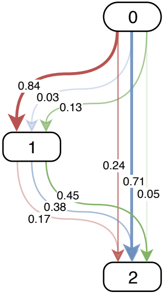
Differentiable Architecture Search (DARTS) [Liu et al. '19]

Continuous Relaxation

- Between 2 nodes (features maps): Categorical choice for which operation to use.
  - Relax discrete space using a convex combination of these choices $\longrightarrow$ supernetwork with shared weights between architectures
- **MixedOp**: $x^{(j)} = \sum_{i<j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i<j} \sum_{o \in \mathcal{O}} \frac{e^{\lambda_o^{(i,j)}}}{\sum_{o' \in \mathcal{O}} e^{\lambda_{o'}^{(i,j)}}} o(x^{(i)})$



(b) Search start

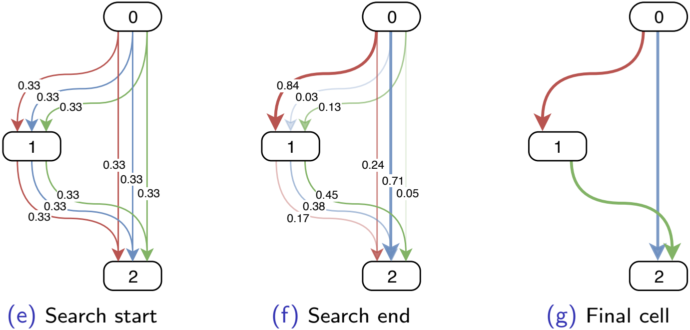


(c) Search end

Differentiable Architecture Search (DARTS) [Liu et al. '19]

Continuous Relaxation

- Between 2 nodes (features maps): Categorical choice for which operation to use.
  - Relax discrete space using a convex combination of these choices $\longrightarrow$ supernetwork with shared weights between architectures
- **MixedOp**: $x^{(j)} = \sum_{i<j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i<j} \sum_{o \in \mathcal{O}} \frac{e^{\lambda_o^{(i,j)}}}{\sum_{o' \in \mathcal{O}} e^{\lambda_{o'}^{(i,j)}}} o(x^{(i)})$
- Disctretize back by keeping the operation with the highest λ



DARTS: Architecture Optimization

Assume we have a discrete architecture space $\mathcal{A}$, a loss function $\mathcal{L}$, and a dataset $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{valid}$.

- $\mathcal{L}$ is a (stochastic) function of both $\lambda \in \mathcal{A}$ and $\mathbf{w} \in \mathbb{R}^d$
- Optimizing both $\mathcal{L}_T := \mathcal{L}(\cdot; \mathcal{D}_{train})$ and $\mathcal{L}_V := \mathcal{L}(\cdot; \mathcal{D}_{valid})$ corresponds to a bi-level optimization problem:

$$\min_{\lambda} \{\mathcal{L}_V^*(\lambda) := \mathcal{L}_V(\mathbf{w}^*(\lambda), \lambda)\} \qquad \text{(upper-level)}$$

$$s.t. \quad \mathbf{w}^*(\lambda) = \arg\min_{\mathbf{w}} \mathcal{L}_{train}(\mathbf{w}, \lambda), \qquad \text{(lower-level)}$$

DARTS: Architecture Optimization

Assume we have a discrete architecture space $\mathcal{A}$, a loss function $\mathcal{L}$, and a dataset $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{valid}$.

- $\mathcal{L}$ is a (stochastic) function of both $\lambda \in \mathcal{A}$ and $\mathbf{w} \in \mathbb{R}^d$
- Optimizing both $\mathcal{L}_T := \mathcal{L}(\cdot; \mathcal{D}_{train})$ and $\mathcal{L}_V := \mathcal{L}(\cdot; \mathcal{D}_{valid})$ corresponds to a bi-level optimization problem:

$$\min_{\lambda} \{\mathcal{L}_V^*(\lambda) := \mathcal{L}_V(\mathbf{w}^*(\lambda), \lambda)\} \qquad \text{(upper-level)}$$
$$s.t. \quad \mathbf{w}^*(\lambda) = \arg\min_{\mathbf{w}} \mathcal{L}_{train}(\mathbf{w}, \lambda), \qquad \text{(lower-level)}$$

- Approximate $\mathbf{w}^*(\lambda) \approx \mathbf{w} - \xi_1 \nabla_{\mathbf{w}} \mathcal{L}_T(\mathbf{w}, \lambda)$
- The optimization alternates between:
  1. $\mathbf{w} \leftarrow \mathbf{w} - \xi_1 \nabla_{\mathbf{w}} \mathcal{L}_T(\mathbf{w}, \lambda)$
  2. $\lambda \leftarrow \lambda - \xi_2 \nabla_{\lambda} \mathcal{L}_V^*(\lambda)$

DARTS: Architecture Optimization

Assume we have a discrete architecture space $\mathcal{A}$, a loss function $\mathcal{L}$, and a dataset $\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{valid}$.

- $\mathcal{L}$ is a (stochastic) function of both $\lambda \in \mathcal{A}$ and $\mathbf{w} \in \mathbb{R}^d$
- Optimizing both $\mathcal{L}_T := \mathcal{L}(\cdot; \mathcal{D}_{train})$ and $\mathcal{L}_V := \mathcal{L}(\cdot; \mathcal{D}_{valid})$ corresponds to a bi-level optimization problem:

$$\min_{\lambda} \{\mathcal{L}_V^*(\lambda) := \mathcal{L}_V(\mathbf{w}^*(\lambda), \lambda)\} \qquad \text{(upper-level)}$$

$$s.t. \quad \mathbf{w}^*(\lambda) = \arg\min_{\mathbf{w}} \mathcal{L}_{train}(\mathbf{w}, \lambda), \qquad \text{(lower-level)}$$

- Approximate $\mathbf{w}^*(\lambda) \approx \mathbf{w} - \xi_1 \nabla_{\mathbf{w}} \mathcal{L}_T(\mathbf{w}, \lambda)$
- The optimization alternates between:
  1. $\mathbf{w} \leftarrow \mathbf{w} - \xi_1 \nabla_{\mathbf{w}} \mathcal{L}_T(\mathbf{w}, \lambda)$
  2. $\lambda \leftarrow \lambda - \xi_2 \nabla_{\lambda} \mathcal{L}_V^*(\lambda)$

where, $\nabla_{\lambda} \mathcal{L}_V^*(\lambda) \approx \nabla_{\lambda} \mathcal{L}_V(\mathbf{w}^*, \lambda) - \xi_1 \nabla_{\mathbf{w}} \mathcal{L}_V(\mathbf{w}^*, \lambda) \nabla^2_{\mathbf{w},\lambda} \mathcal{L}_T(\mathbf{w}^*, \lambda)$

Generalization after discretization

Works quite well on many search spaces

- Original CNN space: 8 operations choices between pairs of nodes
- 28 MixedOPs in total
- $> 10^{10}$ possible architectures
- $< 3\%$ on CIFAR-10 in less than 1 GPU day of search

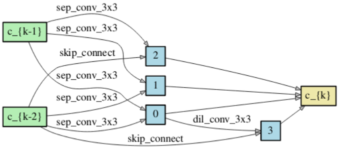


Figure 4: Normal cell learned on CIFAR-10.

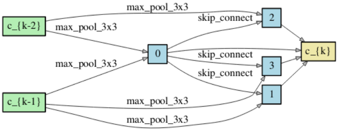


Figure 5: Reduction cell learned on CIFAR-10.

Generalization after discretization

But not always...

S1: This search space uses a different set of two operators per edge.

S2: $\{3 \times 3$ *SepConv*, *SkipConnect*$\}$.

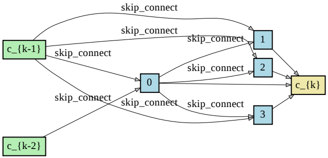
S3: $\{3 \times 3$ *SepConv*, *SkipConnect*, *Zero*$\}$,

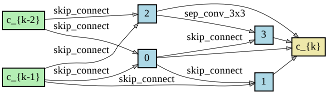
S4: $\{3 \times 3$ *SepConv*, *Noise*$\}$.

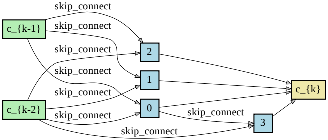
Generalization after discretization

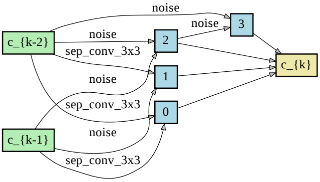
But not always...

- **S1:** This search space uses a different set of two operators per edge.
- **S2:** $\{3 \times 3$ *SepConv*, *SkipConnect*$\}$.
- **S3:** $\{3 \times 3$ *SepConv*, *SkipConnect*, *Zero*$\}$,
- **S4:** $\{3 \times 3$ *SepConv*, *Noise*$\}$.









Generalization after discretization

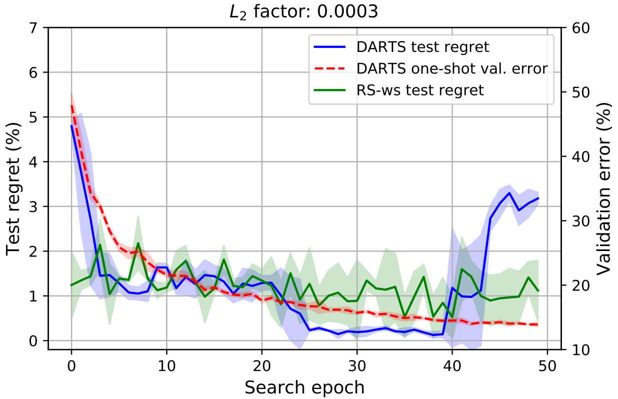
Small toy search space

S5: Very small search space with known global optimum.

- 81 possible architectures trained 3 independent times using the default DARTS settings.

S5: Very small search space with known global optimum.

- 81 possible architectures trained 3 independent times using the default DARTS settings.
- Test performance after discretization **diverges**.



Loss landscape

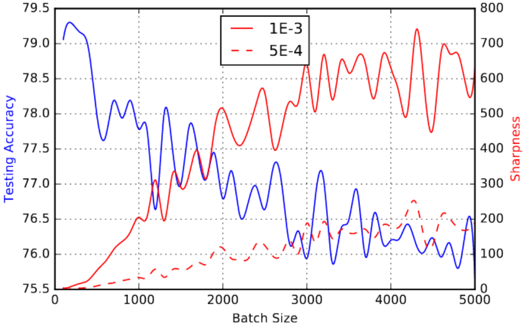
Sharp vs. flat minima

- What is an indicator that the found solutions generalize after discretization?

Loss landscape

Sharp vs. flat minima

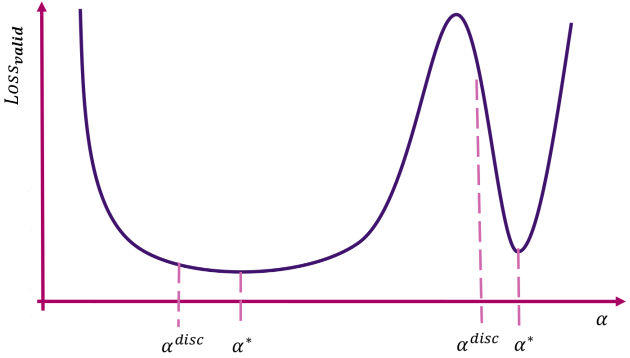
- What is an indicator that the found solutions generalize after discretization?
- Flatness/sharpness of minima, e.g. in large vs. small batch size training of NN, is a good indicator of generalization.



Keskar et al. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In ICLR 2017

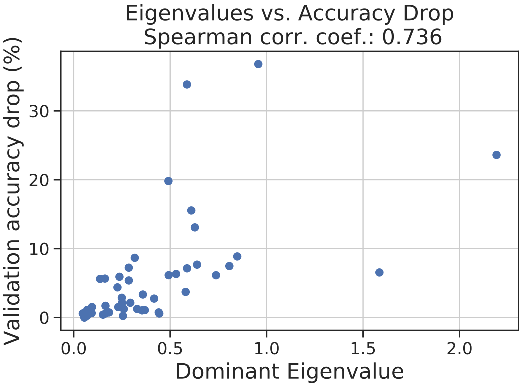
How the curvature relates with generalization?

- Sharp minima much more sensitive to variations in the input space after discretization.
- Discretization via **argmax**: $\lambda_o^* = [3.2, 1.1, 2.5] \longrightarrow \lambda_o^{disc} = [1, 0, 0]$.



How the curvature relates with generalization?

- Sharp minima much more sensitive to variations in the input space after discretization.
- Discretization via **argmax**: $\lambda_o^* = [3.2, 1.1, 2.5] \longrightarrow \lambda_o^{disc} = [1, 0, 0]$.

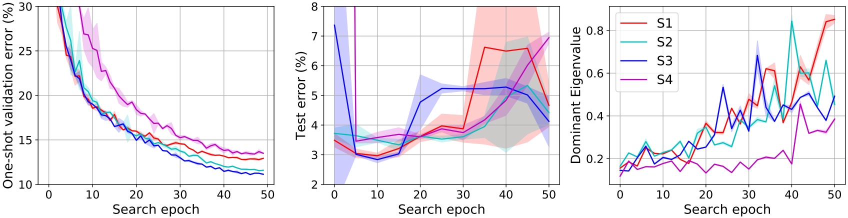


Generalization of architectures and sharpness of minimas

- Compute the full Hessian $\nabla^2_\lambda \mathcal{L}_V$ on a randomly sampled mini-batch from the validation set.

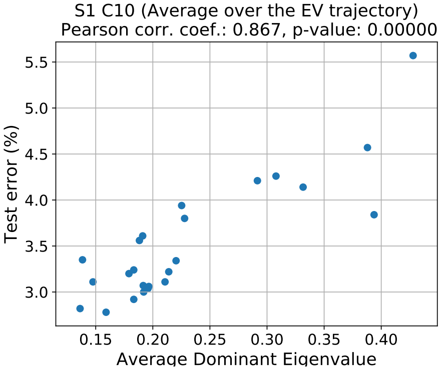
Generalization of architectures and sharpness of minimas

- Compute the full Hessian $\nabla^2_\lambda \mathcal{L}_V$ on a randomly sampled mini-batch from the validation set.
- The dominant EV starts increasing at the point where the architecture generalization error starts increasing.



Generalization of architectures and sharpness of minimas

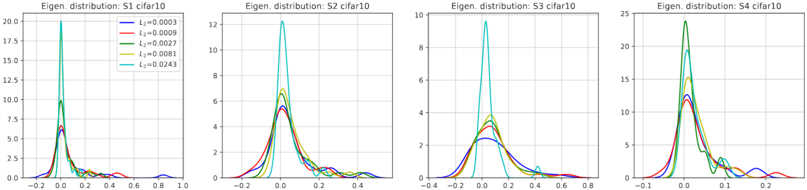
- Compute the full Hessian $\nabla^2_\lambda \mathcal{L}_V$ on a randomly sampled mini-batch from the validation set.
- The dominant EV starts increasing at the point where the architecture generalization error starts increasing.
- High correlation between generalization and the dominant eigenvalue (EV).

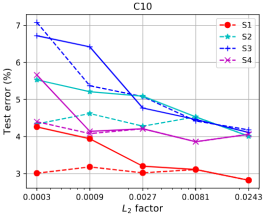


Regularizing the Lower-level Problem

Improved generalization

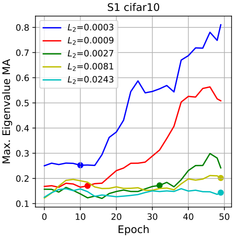
- If lower-level problem strongly convex, convergence of upper-level guaranteed under suitable step size conditions and smoothness.
- Increasing the L_2 regularization in the lower-level $\longrightarrow$ narrower and more convex basins towards $\mathbf{w}$ + smoother landscape.





Goal: Keep the dominant eigenvalue to a low value (or at least return a solution before it increases). If the $\lambda_{max}(i-k)/\lambda_{max}(i) < 3/4$:

- Early stop.
- Roll back and increase regularization.



Setting		RandomNAS	DARTS	DARTS-ES	DARTS-ADA
C10	S1	3.17 ± 0.15	4.66 ± 0.71	3.05 ± 0.07	$\mathbf{3.03} \pm 0.08$
	S2	3.46 ± 0.15	4.42 ± 0.40	$\mathbf{3.41} \pm 0.14$	3.59 ± 0.31
	S3	$\mathbf{2.92} \pm 0.04$	4.12 ± 0.85	3.71 ± 1.14	2.99 ± 0.34
	S4	89.39 ± 0.84	6.95 ± 0.18	4.17 ± 0.21	$\mathbf{3.89} \pm 0.67$
C100	S1	25.81 ± 0.39	29.93 ± 0.41	28.90 ± 0.81	$\mathbf{24.94} \pm 0.81$
	S2	$\mathbf{22.88} \pm 0.16$	28.75 ± 0.92	24.68 ± 1.43	26.88 ± 1.11
	S3	24.58 ± 0.61	29.01 ± 0.24	26.99 ± 1.79	$\mathbf{24.55} \pm 0.63$
	S4	30.01 ± 1.52	24.77 ± 1.51	23.90 ± 2.01	$\mathbf{23.66} \pm 0.90$
SVHN	S1	2.64 ± 0.09	9.88 ± 5.50	2.80 ± 0.09	$\mathbf{2.59} \pm 0.07$
	S2	$\mathbf{2.57} \pm 0.04$	3.69 ± 0.12	2.68 ± 0.18	2.79 ± 0.22
	S3	2.89 ± 0.09	4.00 ± 1.01	2.78 ± 0.29	$\mathbf{2.58} \pm 0.07$
	S4	3.42 ± 0.04	2.90 ± 0.02	2.55 ± 0.15	$\mathbf{2.52} \pm 0.06$

Heuristics

Early Stopping and Adaptive Regularization

Goal: Keep the dominant eigenvalue to a low value (or at least return a solution before it increases). If the $\lambda_{max}(i-k)/\lambda_{max}(i) < 3/4$:

- Early stop.
- Roll back and increase regularization.

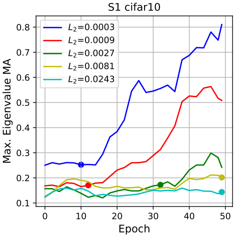


Table 2: Effect of regularization for disparity estimation. Search was conducted on FlyingThings3D (FT) and then evaluated on both FT and Sintel. Lower is better.

Aug. Scale	Search model valid EPE	FT test EPE	Sintel test EPE	Params (M)
0.0	4.49	3.83	5.69	9.65
0.1	3.53	3.75	5.97	9.65
0.5	3.28	3.37	5.22	9.43
1.0	4.61	3.12	5.47	12.46
1.5	5.23	2.60	4.15	12.57
2.0	7.45	**2.33**	**3.76**	12.25
L_2 reg. factor	**Search model valid EPE**	**FT test EPE**	**Sintel test EPE**	**Params (M)**
3×10^{-4}	3.95	3.25	6.13	11.00
9×10^{-4}	5.97	**2.30**	4.12	13.92
27×10^{-4}	4.25	2.72	4.83	10.29
81×10^{-4}	4.61	**2.34**	**3.85**	12.16

Neural Ensemble Search

Quantifying Uncertainty in Deep Learning

Why are they important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification

Quantifying Uncertainty in Deep Learning

Why are they important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification
- Good uncertainty estimates quantify how much we **can trust our model's predictions**

Quantifying Uncertainty in Deep Learning

Why are they important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification
- Good uncertainty estimates quantify how much we **can trust our model's predictions**
- Some applications where uncertainty quantification is important are:

Quantifying Uncertainty in Deep Learning

Why are they important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification
- Good uncertainty estimates quantify how much we **can trust our model's predictions**
- Some applications where uncertainty quantification is important are:
  - Cost-sensitive decision making (healthcare e.g. medical imaging; self-driving cars; robotics)

Quantifying Uncertainty in Deep Learning

Why are they important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification
- Good uncertainty estimates quantify how much we **can trust our model's predictions**
- Some applications where uncertainty quantification is important are:
  - Cost-sensitive decision making (healthcare e.g. medical imaging; self-driving cars; robotics)
  - Dealing with distribution shift (Feature skew between train and test sets; test inputs do not belong to any of the training classes)

Quantifying Uncertainty in Deep Learning

Why are they important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification
- Good uncertainty estimates quantify how much we **can trust our model's predictions**
- Some applications where uncertainty quantification is important are:
  - Cost-sensitive decision making (healthcare e.g. medical imaging; self-driving cars; robotics)
  - Dealing with distribution shift (Feature skew between train and test sets; test inputs do not belong to any of the training classes)
  - Safe exploration in RL, etc.

Quantifying Uncertainty in Deep Learning

Why are they important?

- **Predictive uncertainty** can be for instance the output label together with the confidence of that prediction in classification
- Good uncertainty estimates quantify how much we **can trust our model's predictions**
- Some applications where uncertainty quantification is important are:
  - Cost-sensitive decision making (healthcare e.g. medical imaging; self-driving cars; robotics)
  - Dealing with distribution shift (Feature skew between train and test sets; test inputs do not belong to any of the training classes)
  - Safe exploration in RL, etc.
- Ideally we want a system that knows what it doesn't know.

Calibration and Robustness to dataset shift

Are deep neural networks calibrated and robust to OOD data?

- Calibration tells us how well the predicted confidence (*probability of correctness*) of the model aligns with the observed accuracy (*frequency of correctness*).

Calibration and Robustness to dataset shift

Are deep neural networks calibrated and robust to OOD data?

- Calibration tells us how well the predicted confidence (*probability of correctness*) of the model aligns with the observed accuracy (*frequency of correctness*).
- E.g. in image classification: if the correct predicted class was always with 80% probability, then a perfectly calibrated system would imply that on 80% of the examples it predicted the true class.

Calibration and Robustness to dataset shift

Are deep neural networks calibrated and robust to OOD data?

- Calibration tells us how well the predicted confidence (*probability of correctness*) of the model aligns with the observed accuracy (*frequency of correctness*).
- E.g. in image classification: if the correct predicted class was always with 80% probability, then a perfectly calibrated system would imply that on 80% of the examples it predicted the true class.
- Usually neural networks are **not well-calibrated** making overconfident or underconfident predictions

Calibration and Robustness to dataset shift

Are deep neural networks calibrated and robust to OOD data?

- Calibration tells us how well the predicted confidence (*probability of correctness*) of the model aligns with the observed accuracy (*frequency of correctness*).
- E.g. in image classification: if the correct predicted class was always with 80% probability, then a perfectly calibrated system would imply that on 80% of the examples it predicted the true class.
- Usually neural networks are **not well-calibrated** making overconfident or underconfident predictions
- Moreover, they are fragile, i.e. **they do not have high uncertainty** on out-of-distribution (OOD) inputs.

- *Ensembles* of networks are commonly used to boost performance.

- *Ensembles* of networks are commonly used to boost performance.
- Recent interest in ensembles has been due to their strong *predictive uncertainty estimation* and *robustness to distributional shift*.

- *Ensembles* of networks are commonly used to boost performance.
- Recent interest in ensembles has been due to their strong *predictive uncertainty estimation* and *robustness to distributional shift*.

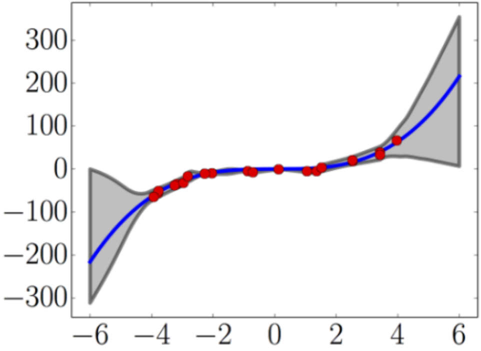


Figure: (from Lakshminarayanan et al. 2017)

Ensembles of Neural Networks

Deep Ensembles

- *Ensembles* of networks are commonly used to boost performance.
- Recent interest in ensembles has been due to their strong *predictive uncertainty estimation* and *robustness to distributional shift*.
- Deep ensembles [Lakshminarayanan et al. 2017], only ensemble predictions coming from the same *fixed* architecture as follows:
  1. Independently train multiple copies of a fixed architecture with random initializations.
  2. Create an ensemble by averaging outputs, i.e. predicted distribution over the classes (in the case of classification).

Ensembles of Neural Networks

Deep Ensembles

- *Ensembles* of networks are commonly used to boost performance.
- Recent interest in ensembles has been due to their strong *predictive uncertainty estimation* and *robustness to distributional shift*.
- Deep ensembles [Lakshminarayanan et al. 2017], only ensemble predictions coming from the same *fixed* architecture as follows:
  1. Independently train multiple copies of a fixed architecture with random initializations.
  2. Create an ensemble by averaging outputs, i.e. predicted distribution over the classes (in the case of classification).
- Diversity among the *base learners'* predictions is believed to be key for strong ensembles.

Ensembles of neural networks

On diversity in ensembles

- Notation: f_θ is a network with weights θ, and $\ell(f_\theta(\boldsymbol{x}), y)$ is the loss for $(\boldsymbol{x}, y)$. Define the ensemble of M networks $f_{\theta_1}, \ldots, f_{\theta_M}$ by $F(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} f_{\theta_i}(\boldsymbol{x})$.

- Notation: f_θ is a network with weights θ, and $\ell(f_\theta(\boldsymbol{x}), y)$ is the loss for $(\boldsymbol{x}, y)$. Define the ensemble of M networks $f_{\theta_1}, \ldots, f_{\theta_M}$ by $F(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} f_{\theta_i}(\boldsymbol{x})$.
- **Average base learner loss:** $\frac{1}{M}\sum_{i=1}^{M} \ell(f_{\theta_i}(\boldsymbol{x}), y)$.

- Notation: f_θ is a network with weights θ, and $\ell(f_\theta(\boldsymbol{x}), y)$ is the loss for $(\boldsymbol{x}, y)$. Define the ensemble of M networks $f_{\theta_1}, \dots, f_{\theta_M}$ by $F(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} f_{\theta_i}(\boldsymbol{x})$.
- **Average base learner loss:** $\frac{1}{M}\sum_{i=1}^{M} \ell(f_{\theta_i}(\boldsymbol{x}), y)$.
- **Oracle ensemble:** given $f_{\theta_1}, \dots, f_{\theta_M}$, the oracle ensemble F_{OE} is defined as

$$F_{\mathsf{OE}}(\boldsymbol{x}) = f_{\theta_k}(\boldsymbol{x}), \quad \text{where} \quad k \in \arg\min_i \ell(f_{\theta_i}(\boldsymbol{x}), y).$$

Ensembles of neural networks

On diversity in ensembles

- Notation: f_θ is a network with weights θ, and $\ell(f_\theta(\boldsymbol{x}), y)$ is the loss for $(\boldsymbol{x}, y)$. Define the ensemble of M networks $f_{\theta_1}, \dots, f_{\theta_M}$ by $F(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} f_{\theta_i}(\boldsymbol{x})$.
- **Average base learner loss:** $\frac{1}{M}\sum_{i=1}^{M} \ell(f_{\theta_i}(\boldsymbol{x}), y)$.
- **Oracle ensemble:** given $f_{\theta_1}, \dots, f_{\theta_M}$, the oracle ensemble F_{OE} is defined as

$$F_{\text{OE}}(\boldsymbol{x}) = f_{\theta_k}(\boldsymbol{x}), \quad \text{where} \quad k \in \arg\min_i \ell(f_{\theta_i}(\boldsymbol{x}), y).$$

- As a rule of thumb, *small oracle ensemble loss indicates more diverse base learner predictions*.

Proposition

Suppose ℓ is negative log-likelihood (NLL). Then, the oracle ensemble loss, ensemble loss, and average base learner loss satisfy the following inequality:

$$\ell(F_{OE}(\boldsymbol{x}), y) \leq \ell(F(\boldsymbol{x}), y) \leq \frac{1}{M} \sum_{i=1}^{M} \ell(f_{\theta_i}(\boldsymbol{x}), y).$$

Proof.

Taking $\ell(f(\boldsymbol{x}), y)) = -\log\,[f(\boldsymbol{x})]_y$ (convex function), it follows direct by applying Jensen's inequality for the right inequality and definition of oracle ensemble for the left one.

$[f(\boldsymbol{x})]_y$ is the probability assigned by the network f of $\boldsymbol{x}$ belonging to the true class y.

Varying vs. fixed base learner architectures

Visualizing base learner predictions using t-SNE on CIFAR-10

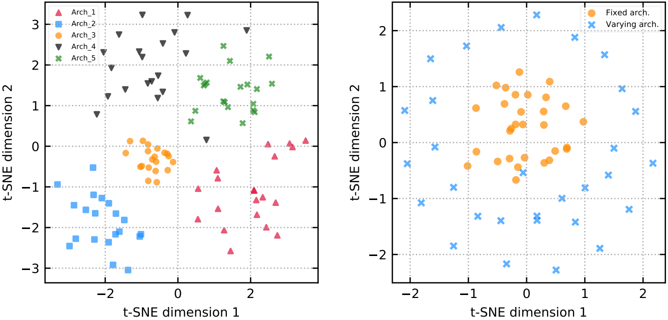


Figure: **Left:** Predictions of 5 different archs, each trained with 20 different inits. **Right:** Predictions of base learners in an ensemble with varying archs (found using NES) vs. fixed arch (deep ensemble of optimized arch).

Neural Ensemble Search

General approach

Let $f_{\theta,\lambda}$ denote a network with arch $\lambda \in \mathcal{A}$ and weights θ. Computational budget denoted by K and ensemble size by M. We want to solve:

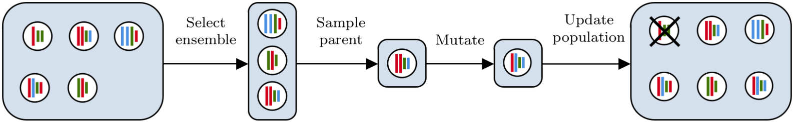
$$\min_{\lambda_1,\ldots,\lambda_M \in \mathcal{A}} \mathcal{L}\left(\texttt{Ensemble}(f_{\theta_1,\lambda_1},\ldots,f_{\theta_M,\lambda_M}), \mathcal{D}_{\text{val}}\right)$$

$$\text{s.t.} \quad \theta_i \in \arg\min_{\theta} \mathcal{L}(f_{\theta,\lambda_i}, \mathcal{D}_{\text{train}}) \qquad \text{for } i = 1,\ldots,M$$

Our approach for finding base learner architectures that optimize ensemble performance consists of two steps.

1. **Pool building**: build a *pool* $= \{f_{\theta_1,\lambda_1},\ldots,f_{\theta_K,\lambda_K}\}$ of size K consisting of potential base learners, where each f_{θ_i,λ_i} is a network trained independently on $\mathcal{D}_{\text{train}}$.
2. **Ensemble selection**: select M base learners $f_{\theta_1^*,\lambda_1^*},\ldots,f_{\theta_M^*,\lambda_M^*}$ from to form an ensemble which minimizes loss on $\mathcal{D}_{\text{val}}$. (We assume $K \geq M$.)

- NES-RS is a simple random search (RS) based approach: we build the pool by sampling K architectures uniformly at random.
- NES-RE uses regularized evolution (RE) [Real et al. 2019] to evolve a population of architectures until the budget K is reached.

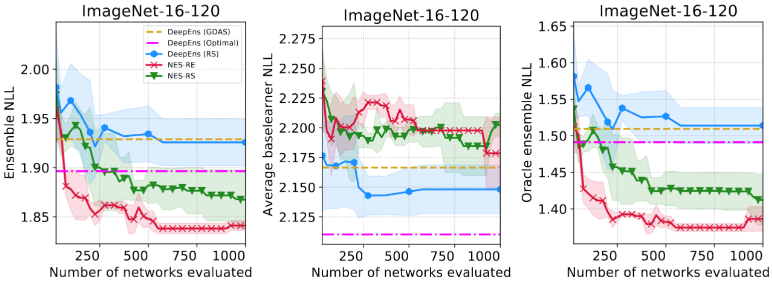


- For step 2, we use forward step-wise selection without replacement: given pool , start with an empty ensemble and add to it the network from which minimizes ensemble loss on $\mathcal{D}_{\text{val}}$. We repeat this without replacement until the ensemble is of size M. (Caruana et al., 2004)

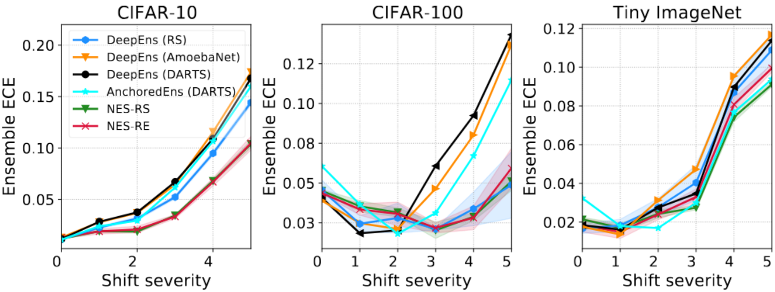
Neural Ensemble Search

Results on common image classification benchmarks

- Results on NAS-Bench-201 [Dong et al. 2020].
  - *Ensembles from NES better than deep ensembles of the global minimum.*



- Results on NAS-Bench-201 [Dong et al. 2020].
  - *Ensembles from NES better than deep ensembles of the global minimum.*
  - Better calibrated on dataset shift.



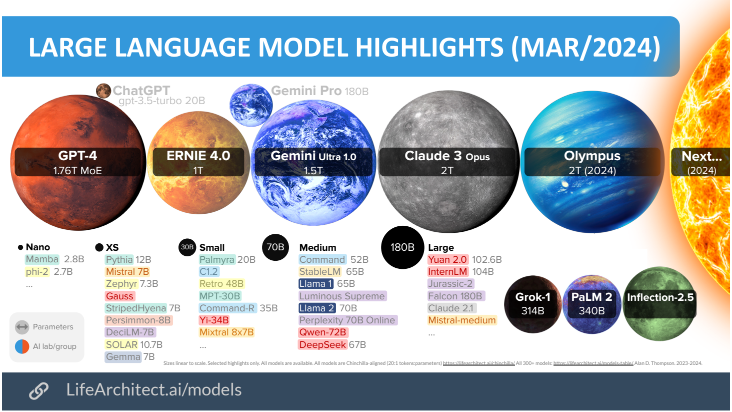
Multi-objective Differentiable Architecture Search

NAS in a world of ever growing models

- In an age of **large models**, finding architectures which are *performant*, *efficient* and with *fast* inference times is pivotal.

NAS in a world of ever growing models

- In an age of **large models**, finding architectures which are *performant*, *efficient* and with *fast* inference times is pivotal.



NAS in a world of ever growing models

- In an age of **large models**, finding architectures which are *performant*, *efficient* and with *fast* inference times is pivotal.
- Multi-objective problem with (potentially) **conflicting objectives**.
    - Optimizing all objectives simultaneously is infeasible.
    - Finding the **right trade-off** remains challenging.

NAS in a world of ever growing models

- In an age of **large models**, finding architectures which are *performant*, *efficient* and with *fast* inference times is pivotal.
- Multi-objective problem with (potentially) **conflicting objectives**.
  - Optimizing all objectives simultaneously is infeasible.
  - Finding the **right trade-off** remains challenging.
- We also need efficient search methods for these kind of spaces.
  - Conventional blackbox methods, such as ES or BO, require multiple expensive evaluations.

Optimality in Multi-objective optimization

Pareto optimality and Pareto front

MOO then seeks to find a set of Pareto-optimal solutions $\alpha^* \in \mathcal{A} \subset \mathbb{R}^d$ that jointly minimize $\mathbf{L}(\alpha) = (\mathcal{L}^1(\alpha), \ldots, \mathcal{L}^M(\alpha))$:

$$\alpha^* \in \arg\min_{\alpha \in \mathcal{A}} \mathbf{L}(\alpha)$$

Optimality in Multi-objective optimization

Pareto optimality and Pareto front

MOO then seeks to find a set of Pareto-optimal solutions $\alpha^* \in \mathcal{A} \subset \mathbb{R}^d$ that jointly minimize $\mathbf{L}(\alpha) = (\mathcal{L}^1(\alpha), \ldots, \mathcal{L}^M(\alpha))$:

$$\alpha^* \in \arg\min_{\alpha \in \mathcal{A}} \mathbf{L}(\alpha)$$

Definition

(**Pareto Optimality**): A solution α_2 dominates α_1 iff $\mathcal{L}^m(\alpha_2) \leq \mathcal{L}^m(\alpha_1)$, $\forall m \in \{1, \ldots, M\}$, and $\mathbf{L}(\alpha_1) \neq \mathbf{L}(\alpha_2)$. In other words, a dominating solution has a lower loss value on at least one task and no higher loss value on any task. A solution α^* is called *Pareto optimal* iff there exists no other solution dominating α^*.

Optimality in Multi-objective optimization

Pareto optimality and Pareto front

MOO then seeks to find a set of Pareto-optimal solutions $\alpha^* \in \mathcal{A} \subset \mathbb{R}^d$ that jointly minimize $\mathbf{L}(\alpha) = (\mathcal{L}^1(\alpha), \ldots, \mathcal{L}^M(\alpha))$:

$$\alpha^* \in \arg\min_{\alpha \in \mathcal{A}} \mathbf{L}(\alpha)$$

Definition

(**Pareto Front**): The sets of Pareto optimal points and their function values are called *Pareto set* ($\mathcal{P}_\alpha$) and *Pareto front* ($\mathcal{P}_\mathbf{L} = \{\mathbf{L}(\alpha)_{\alpha \in \mathcal{P}_\alpha}\}$), respectively.

Definition

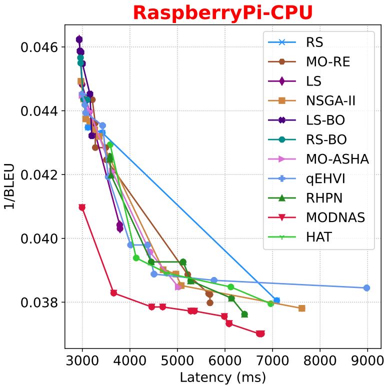
(**Pareto Criticality**): A solution $\alpha^* \in \mathcal{A}$ is called Pareto critical if there is no common descent direction $\boldsymbol{d} \in \mathbb{R}^d$ such that $\nabla \mathcal{L}^i(\alpha^*)^\top \boldsymbol{d} < 0$, $\forall i = 1, \ldots, M$.

Optimality in Multi-objective optimization

Pareto optimality and Pareto front

MOO then seeks to find a set of Pareto-optimal solutions $\alpha^* \in \mathcal{A} \subset \mathbb{R}^d$ that jointly minimize $\mathbf{L}(\alpha) = (\mathcal{L}^1(\alpha), \dots, \mathcal{L}^M(\alpha))$:

$$\alpha^* \in \arg\min_{\alpha \in \mathcal{A}} \mathbf{L}(\alpha)$$



Problem Formulation

Multi-objective NAS as bi-level optimization

Assuming we have **T hardware devices** (target functions) and **M objectives** (e.g. accuracy, latency, energy usage, etc.), the Pareto set $\mathcal{P}_{\alpha_t}$ of the multi-objective NAS problem is obtained by solving the following bi-level optimization problem:

$$\arg\min_{\alpha} \mathbf{L}_t^{valid}(\mathbf{w}^*(\alpha), \alpha)$$
$$s.t. \quad \mathbf{w}^*(\alpha) = \arg\min_{\mathbf{w}} \mathbf{L}_t^{train}(\mathbf{w}, \alpha),$$

where the M-dimensional loss vector $\mathbf{L}_t(\mathbf{w}^*, \alpha) := \big(\mathcal{L}_t^1(\mathbf{w}^*, \alpha), \dots, \mathcal{L}_t^M(\mathbf{w}^*, \alpha)\big)$ is evaluated $\forall t \in \{1, \dots, T\}$.

Problem Formulation

Multi-objective NAS as bi-level optimization

Assuming we have **T hardware devices** (target functions) and **M objectives** (e.g. accuracy, latency, energy usage, etc.), the Pareto set $\mathcal{P}_{\alpha_t}$ of the multi-objective NAS problem is obtained by solving the following bi-level optimization problem:

$$\begin{aligned} &\arg\min_{\alpha} \mathbf{L}_t^{valid}(\mathbf{w}^*(\alpha), \alpha) \\ s.t. \quad &\mathbf{w}^*(\alpha) = \arg\min_{\mathbf{w}} \mathbf{L}_t^{train}(\mathbf{w}, \alpha), \end{aligned}$$

where the M-dimensional loss vector $\mathbf{L}_t(\mathbf{w}^*, \alpha) := \big(\mathcal{L}_t^1(\mathbf{w}^*, \alpha), \ldots, \mathcal{L}_t^M(\mathbf{w}^*, \alpha)\big)$ is evaluated $\forall t \in \{1, \ldots, T\}$.

- Still expensive: Need to run the search T times.

MODNAS

Algorithmic components

1. `MetaPredictor`: regression model to predict cheap-to-evaluate hardware objectives (e.g. latency, energy usage, etc.)
2. `Supernetwork`: proxy to approximate the lower level best response function $\mathbf{w}^*(\alpha)$
3. `MetaHypernetwork`: hypernetwork to generate unnormalized architectural distribution conditioned on preference vectors and hardware device type
4. `Architect`: samples from the architectural distribution discrete architectures

MODNAS

MetaPredictor

- For the cheap-to-evaluate hardware objectives, such as latency, energy consumption.

HELP: Hardware-adaptive Efficient Latency Prediction for NAS via Meta-Learning, Lee et al. NeurIPS 2021

- For the cheap-to-evaluate hardware objectives, such as latency, energy consumption.
- A parametric regression model (e.g. MLP) $p_\theta^m(\alpha, d_t^m) : \mathcal{A} \times \mathcal{H} \rightarrow \mathbb{R}$.

HELP: Hardware-adaptive Efficient Latency Prediction for NAS via Meta-Learning, Lee et al. NeurIPS 2021

- For the cheap-to-evaluate hardware objectives, such as latency, energy consumption.
- A parametric regression model (e.g. MLP) $p_\theta^m(\alpha, d_t^m) : \mathcal{A} \times \mathcal{H} \to \mathbb{R}$.
- Optimize the MSE loss:

$$\min_\theta \mathbb{E}_{\alpha \sim \mathcal{A}, t \sim [T]} \left(y_t^m - p_\theta^m(\alpha, d_t^m)\right)^2$$

HELP: Hardware-adaptive Efficient Latency Prediction for NAS via Meta-Learning, Lee et al. NeurIPS 2021

MODNAS

MetaPredictor

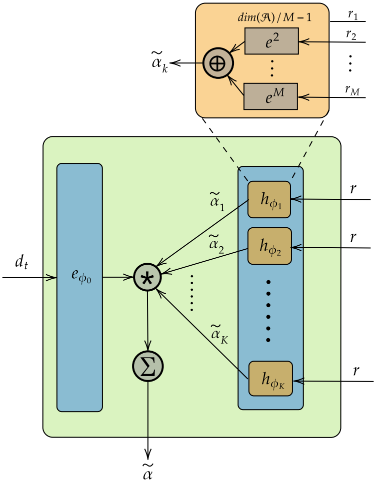
- For the cheap-to-evaluate hardware objectives, such as latency, energy consumption.
- A parametric regression model (e.g. MLP) $p_\theta^m(\alpha, d_t^m) : \mathcal{A} \times \mathcal{H} \to \mathbb{R}$.
- Optimize the MSE loss:

$$\min_\theta \mathbb{E}_{\alpha \sim \mathcal{A}, t \sim [T]} \left(y_t^m - p_\theta^m(\alpha, d_t^m)\right)^2$$

- Use $\mathcal{L}_t^m(\cdot, \alpha_\Phi) = p_\theta^m(\alpha_\Phi, d_t^m)$ as the objective function.
  - During the search we freeze and do not update further the `MetaPredictor` parameters θ.

HELP: Hardware-adaptive Efficient Latency Prediction for NAS via Meta-Learning, Lee et al. NeurIPS 2021

- We use a hypernetwork $H_\Phi(\boldsymbol{r}, d_t)$ that maps a **device embedding** d_t and **preference vector** $\boldsymbol{r} \in \mathbb{R}^M$ to an architecture distribution $\tilde{\alpha}$.
  - Just a forward pass to generate an architecture.
  - Scalable across different hardware devices.



MODNAS
Linear Scalarization

Using the **preference vector** $\boldsymbol{r}$ to create a linear scalarization of $\mathbf{L}_t$ and the `MetaHypernetwork` to model the architectural distribution across T devices, the bi-level problem reduces to:

$$\begin{aligned} &\arg\min_{\Phi} \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\left[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{valid}(\mathbf{w}^*(\alpha_\Phi), \alpha_\Phi)\right] \\ s.t. \quad &\mathbf{w}^*(\alpha_\Phi) = \arg\min_{\mathbf{w}} \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\left[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{train}(\mathbf{w}, \alpha_\Phi)\right], \end{aligned}$$

where $\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t(\cdot, \alpha_\Phi) = \sum_{m=1}^{M} r_m \mathcal{L}_t^m(\cdot, \alpha_\Phi)$ is the scalarized loss for device t.

MODNAS

Linear Scalarization

Using the **preference vector** $\boldsymbol{r}$ to create a linear scalarization of $\mathbf{L}_t$ and the `MetaHypernetwork` to model the architectural distribution across T devices, the bi-level problem reduces to:

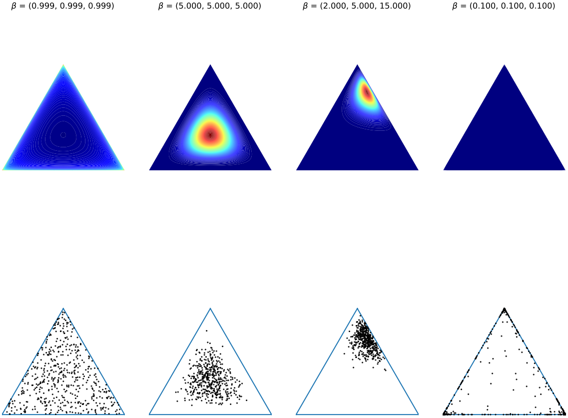
$$\arg\min_{\Phi} \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\left[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{valid}(\mathbf{w}^*(\alpha_\Phi), \alpha_\Phi)\right]$$

$$s.t. \quad \mathbf{w}^*(\alpha_\Phi) = \arg\min_{\mathbf{w}} \mathbb{E}_{\boldsymbol{r}\sim\mathcal{S}}\left[\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t^{train}(\mathbf{w}, \alpha_\Phi)\right],$$

where $\boldsymbol{r}^{\mathbf{T}}\mathbf{L}_t(\cdot, \alpha_\Phi) = \sum_{m=1}^{M} r_m \mathcal{L}_t^m(\cdot, \alpha_\Phi)$ is the scalarized loss for device t.

- Conditioning the `MetaHypernetwork` on the hardware embeddings generates architectures on new test devices without additional update steps.

We sample the preference vector $\boldsymbol{r}$ from a Dirichlet distribution with concentration parameters $\beta_1, \ldots, \beta_M = 1$.



Multiple Gradient Descent (MGD) (*Désidéri 2012*) aims to find a direction $\boldsymbol{d_k}$ that maximizes the minimum decrease across the losses by solving:

$$\max_{\mathbf{d}\in\mathbb{R}^d} \min_{t\in[T]} \left(\mathbf{L}_t(\alpha_k) - \mathbf{L}_(\alpha_k + \eta\mathbf{d}_k)\right) \approx -\eta \min_{\mathbf{d}\in\mathbb{R}^d} \max_{t\in[T]} \nabla\mathbf{L}_t(\alpha_k)^\top \mathbf{d}_k.$$

Regularizing the norm of $\mathbf{d}_k$ on the right hand side and minimizing its dual yields the direction $\mathbf{d}_k$.

MODNAS

Optimizing the `MetaHypernetwork` with MGD

- Multiple Gradient Descent (MGD) (Désidéri 2012) seeks to simultaneously optimize the `MetaHypernetwork` parameters (shared across all devices) $\Phi \leftarrow \Phi - \xi g^*_\Phi$, where: $g^*_\Phi = \sum_{t=1}^{T} \gamma^*_t g^t_\Phi$

- Multiple Gradient Descent (MGD) (Désidéri 2012) seeks to simultaneously optimize the `MetaHypernetwork` parameters (shared across all devices) $\Phi \leftarrow \Phi - \xi g_{\Phi}^{*}$, where: $g_{\Phi}^{*} = \sum_{t=1}^{T} \gamma_t^* g_{\Phi}^{t}$
- Optimal γ_t^*:

$$\min_{\gamma_1,\dots,\gamma_T} \left\{ \left\| \sum_{t=1}^{T} \gamma_t g_{\Phi}^{t} \right\|_2^2 \middle| \sum_{t=1}^{T} \gamma_t = 1, \gamma_t \geq 0, \forall t \right\}.$$

- Multiple Gradient Descent (MGD) (Désidéri 2012) seeks to simultaneously optimize the `MetaHypernetwork` parameters (shared across all devices) $\Phi \leftarrow \Phi - \xi g_\Phi^*$, where: $g_\Phi^* = \sum_{t=1}^{T} \gamma_t^* g_\Phi^t$
- Optimal γ_t^*:

$$\min_{\gamma_1,\dots,\gamma_T} \left\{ \left\| \sum_{t=1}^{T} \gamma_t g_\Phi^t \right\|_2^2 \,\middle|\, \sum_{t=1}^{T} \gamma_t = 1, \gamma_t \geq 0, \forall t \right\}.$$

- $T = 2$:

$$\gamma^* = \max\left(\min\left(\frac{(g_\Phi^2 - g_\Phi^1)^{\mathbf{T}} g_\Phi^2}{\left\|g_\Phi^1 - g_\Phi^2\right\|_2^2}, 1\right), 0\right)$$

- $T > 2$:

  Frank-Wolfe solver `[Jaggi, 2013]`

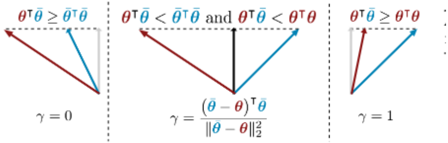


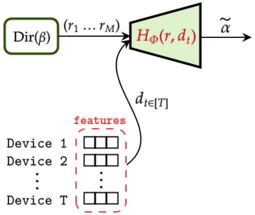
Figure 1: Visualisation of the min-norm point in the convex hull of two points ($\min_{\gamma\in[0,1]} \|\gamma\theta + (1-\gamma)\bar{\theta}\|_2^2$). As the geometry suggests, the solution is either an edge case or a perpendicular vector.

Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization, Jaggi, ICML 2013

Figure from: Multi-Task Learning as Multi-Objective Optimization. Sener and Koltun, NeurIPS 2018

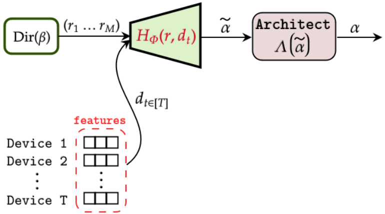
MODNAS

All pieces together



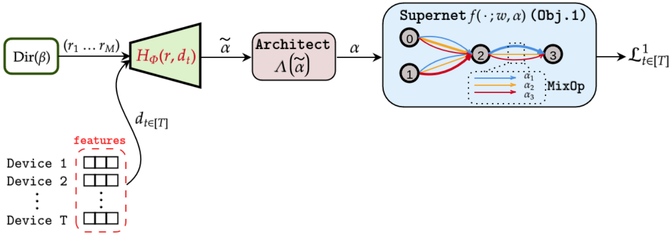
MODNAS

All pieces together



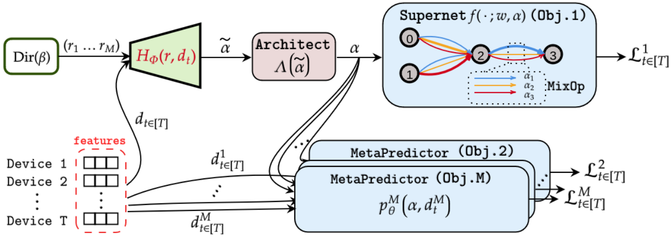
MODNAS

All pieces together



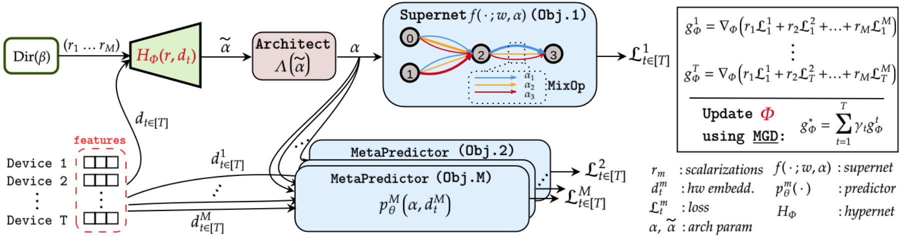
MODNAS

All pieces together



MODNAS

All pieces together



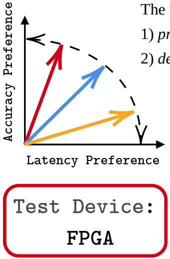
MODNAS

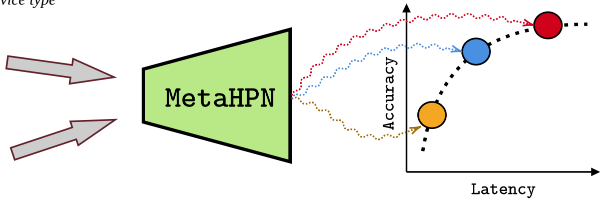
Optimized `MetaHypernetwork`

The trained MetaHPN approximates the Pareto front by generating architectures given:

1) *preference vectors*

2) *device type*

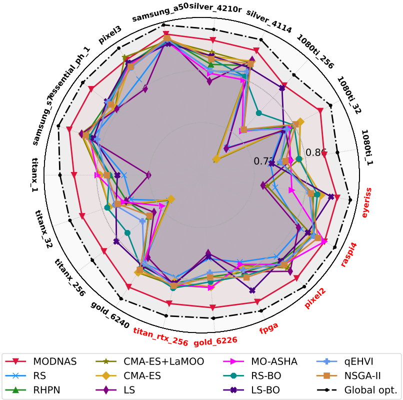




Experimental results

Simultaneous Pareto Set Learning across 19 devices for image classification

- **Metric**: Hypervolume (HV) indicator.
- **Baselines**:
  - Random baselines
  - Evolutionary strategies
  - Bayesian Optimization
- **Evaluation**: Sample 24 preference vectors and get the MAP architecture from the `MetaHypernetwork` output for each of them.



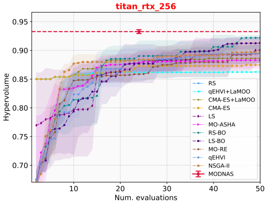
Experimental results

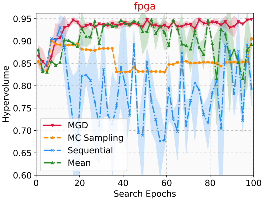
`MetaHypernetwork` update schemes: robustness of MGD

- **Metric**: Hypervolume (HV) indicator over time.

- **Baselines**:
  - Mean grad update
  - Sequential grad updates
  - Grad samples updates

- **Evaluation**: Sample 24 preference vectors and get the MAP architecture from the `MetaHypernetwork` output for each of them.

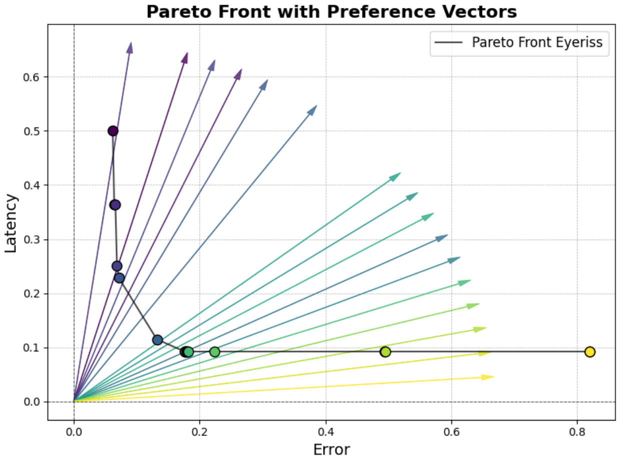




Experimental results

Preference vectors and Pareto front

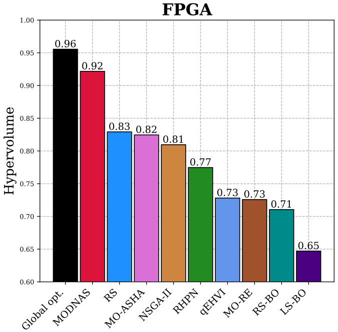
- Preference vectors align relatively well with generated Pareto set image.

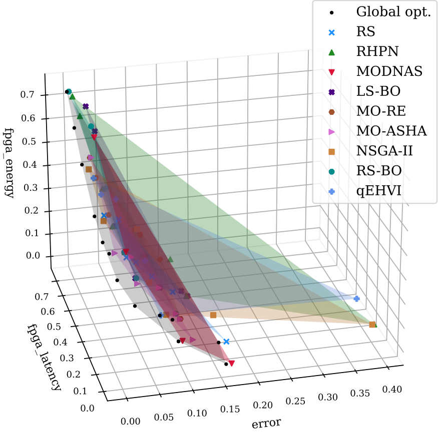


Experimental results

Scalability to 3 objectives

- Optimize for *latency, energy usage* and *accuracy* simultaneously across devices.

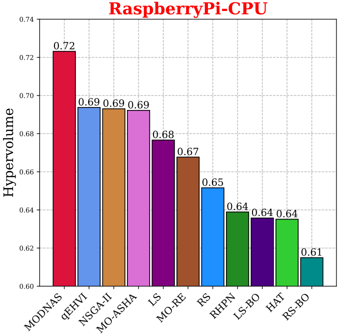


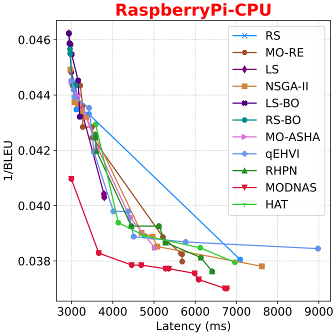


Experimental results

Pareto front profiling on Transformer spaces for machine translation

- Transformer search space on the WMT'14 En-De machine translation task.
- **Search costs:** 6 days on 8 NVidia RTX A6000

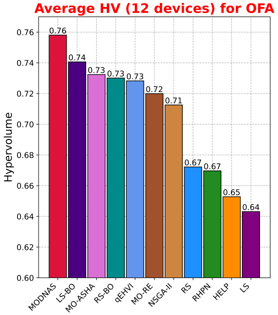


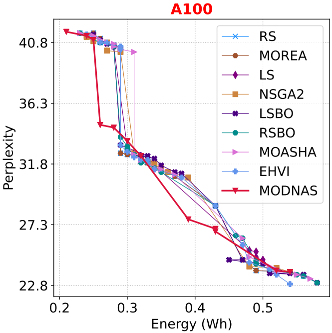


Experimental results

ImageNet-1k and OpenWebText starting from pretrained models

- We use a pretrained model on ImageNet and run MODNAS for 8 GPU days.
- We also evaluate it on a GPT-2 search space.





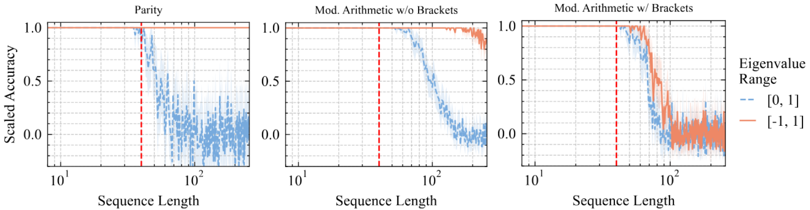
Linear RNNs for State-tracking

Linear RNNs solve Parity and Mod Arithmetic

Theorem (Parity)

A finite precision LRNN with finitely many layers as in (1) can solve parity for arbitrary input lengths, in particular, it can recognize the language $(11)^*$*, only if in at least one layer, there exist* $\boldsymbol{x}$ *such that* $\boldsymbol{A}(\boldsymbol{x})$ *has at least one eigenvalue* λ *with* $|\lambda| \geq 1$ *and* $\lambda \notin \{x \in : x \geq 0\}$.

$$\begin{aligned} &\boldsymbol{H}_i = \boldsymbol{A}(\boldsymbol{x}_i)\boldsymbol{H}_{i-1} + \boldsymbol{B}(\boldsymbol{x}_i), \quad \hat{\boldsymbol{y}}_i = \mathrm{dec}(\boldsymbol{H}_i, \boldsymbol{x}_i), \quad \text{for all } i \in \{1, \dots, t\}, \\ &\boldsymbol{H}_0 \in \mathbb{C}^{n \times d}, \quad \boldsymbol{A} : \mathbb{R}^l \to \mathbb{C}^{n \times n}, \quad \boldsymbol{B} : \mathbb{R}^l \to \mathbb{C}^{n \times d}, \quad \mathrm{dec} : \mathbb{C}^{n \times d} \times \mathbb{R}^l \to \mathbb{R}^p \end{aligned} \tag{1}$$



Conclusions and Meta-Remarks

Conclusions & Meta-Remarks

- **Optimization and Generalization:** Loss landscape in bi-level optimization.
- **Efficiency**: Multi-objective optimization in architecture spaces.
- **Robustness**: Better uncertainty estimation and calibration via NES.
- **Expressive power:** Linear RNNs for state tracking tasks.

Conclusions & Meta-Remarks

- **Optimization and Generalization:** Loss landscape in bi-level optimization.
- **Efficiency**: Multi-objective optimization in architecture spaces.
- **Robustness**: Better uncertainty estimation and calibration via NES.
- **Expressive power:** Linear RNNs for state tracking tasks.

- Why do I want to join your group?
  1. I would like to better understand how such algorithms work.
  2. I want to get better in theory, and I think the best and fastest way to do that is to work with people who are better at it.
  3. I want to stay in academia.

Thank you for your attention. Questions?