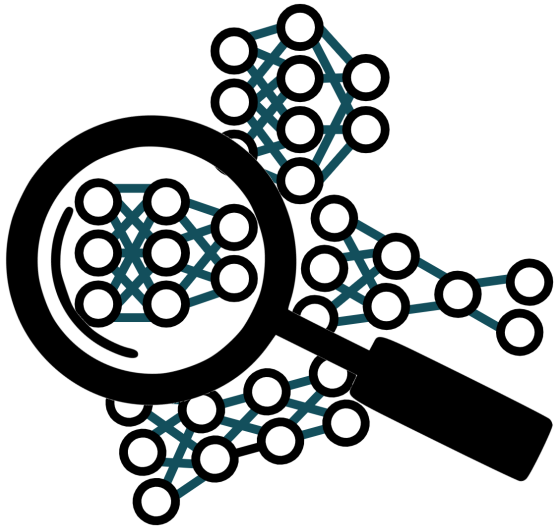
Steepest Descent Neural Architecture Optimization: Going Beyond Black Boxes

Qiang Liu
UT Austin

December 20, 2019

Neural Architecture Optimization in Deep Learning

- Neural architecture optimization is **of critical importance**:
  - Significantly improving the **accuracy** (beyond parameter optimization).
  - Enabling automatic machine learning (**AutoML**).
  - Search **computation/energy-efficient** architectures for mobile, IoT settings.



Neural Architecture Optimization in Deep Learning

Parameter learning has been found "easy" via gradient-based optimization (a.k.a. back-propagation).

But neural architecture optimization is much more difficult...

- Discrete combinatorial optimization:
  - *large search space*
  - *expensive evaluation*
- Mostly solved by *derive-free, brute-force, black-box optimizers*:
  - Evolutionary, genetic algorithms, reinforcement learning, etc.
  - Requires expensive computational resource to succeed.
- Theoretical / mathematical studies have been largely missing!

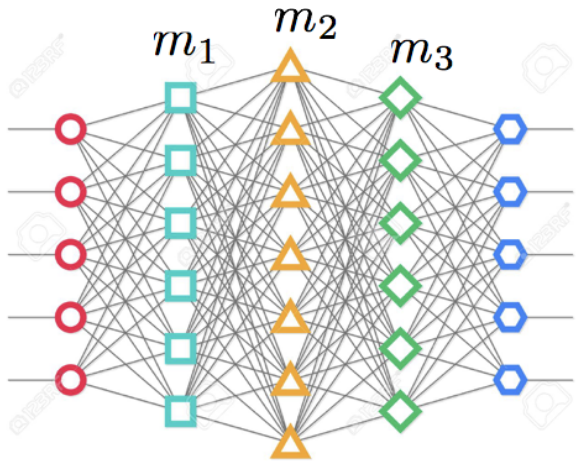
Question: Can we derive fast "gradient-descent-like" algorithms for neural architecture optimization?

- **This work**:
  - A steepest descent approach for neural architecture optimization.
  - A practical algorithm that progressively grows networks by "splitting neurons".
  - Fast and practical, learns accurate and compact neural architectures.

Overall Framework

- Structure of a d-layer DNN is characterized by the width of each layers

$$\boldsymbol{m} = [m_1, m_2, \ldots, m_d].$$

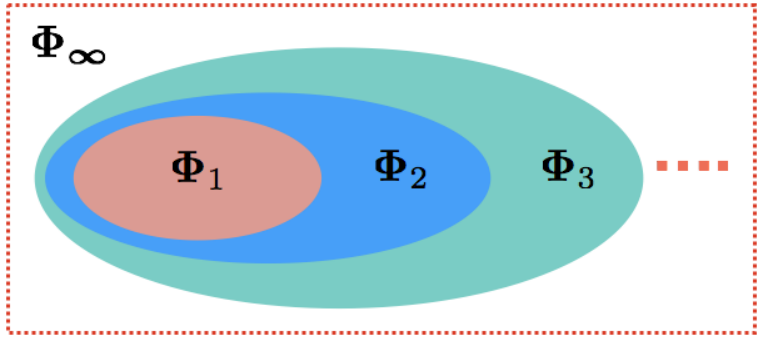


- Let $\boldsymbol{\Theta_m}$ be the space of parameters with structure $\boldsymbol{m}$.
- the Overall model space is

$$\boldsymbol{\Theta_\infty} = \cup_{\boldsymbol{m}\in\mathbb{N}^d} \boldsymbol{\Theta_m}.$$

- Structure-Parameter Co-optimization:

$$\min_{\theta\in\boldsymbol{\Theta_\infty}} L(\theta)$$



- Yields (infinite dimensional) continuous optimization, solve it by (functional) steepest descent!

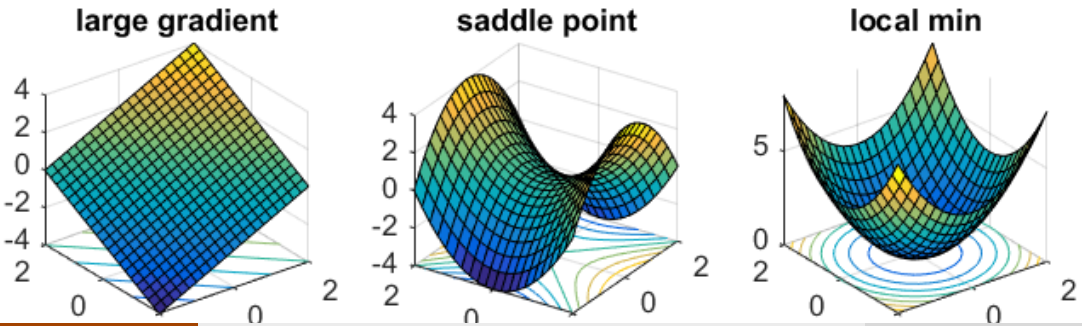
Steepest Descent: Parametric Case

Recall parametric steepest descent on $\mathbb{R}^d$:

$$\boldsymbol{\theta}_{t+1} \asymp \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \left\{ L(\boldsymbol{\theta}) - L(\boldsymbol{\theta}_t) \quad s.t. \quad \|\boldsymbol{\theta} - \boldsymbol{\theta}_t\| \leq \epsilon \right\},$$

When $\|\cdot\|$ is Euclidean norm and $\epsilon \to 0^+$, reduces to

- gradient descent at non-stationary points.
- "eigen-descent" at saddle points or local maxima.
  - naturally escaped by stochastic gradient descent.
- convergence at local minima.



We Want to Derive Steepest Descent on Θ_∞

- Equip $\boldsymbol{\Theta}_\infty$ with a proper notion of distance $\mathbb{D}(\boldsymbol{\theta},\ \boldsymbol{\theta}')$.
- Derive steepest descent on $\boldsymbol{\Theta}_\infty$:

$$\boldsymbol{\theta}_{t+1} \asymp \arg\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}_\infty} \left\{ L(\boldsymbol{\theta}) - L(\boldsymbol{\theta}_t) \quad s.t.\ \ \mathbb{D}(\boldsymbol{\theta},\ \boldsymbol{\theta}_t) \le \epsilon \right\},$$

  - ϵ: small step size.
  - Hope to derive simple update formula when $\epsilon \to 0^+$.

Key Challenge: How to define distance $\mathbb{D}(\cdot,\cdot)$ on Θ_∞?

- Measures inherent difference of DNNs with **different sizes**.
- Yields fast and practical algorithms.

Use Wasserstein Metrics!

- Idea: Match the sizes using optimal transport.
- Consider two one-hidden-layer neural networks of different sizes:

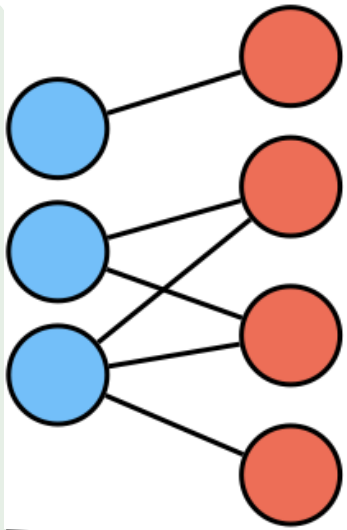
$$f(x;\ \boldsymbol{\theta}) = \sum_{i=1}^{m} w_i \sigma(\theta_i, x), \qquad f(x;\ \boldsymbol{\theta}') = \sum_{i=1}^{m'} w_i' \sigma(\theta_i', x),$$

where $\boldsymbol{\theta} = \{w_i, \theta_i\}_{i=1}^{m}$ and $\boldsymbol{\theta}' = \{w_i', \theta_i'\}_{i=1}^{m'}$ are of different sizes.

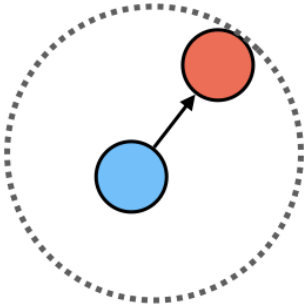
∞-Wasserstein metric:

$$\mathbb{D}_\infty(\boldsymbol{\theta},\ \boldsymbol{\theta}') = \inf_{\gamma \in \Gamma} \max_{ij:\ \gamma_{ij} \neq 0} \left\| \theta_i - \theta_j' \right\|$$

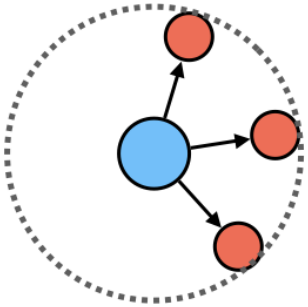
Γ: the set of $m \times m'$ matrices with $\gamma_{ij} \geq 0$, $\sum_j \gamma_{ij} = w_i$ and $\sum_i \gamma_{ij} = w_j'$, for all $\forall i, j$. Assume $\sum_i w_i = 1$ and $w_i \geq 0$.



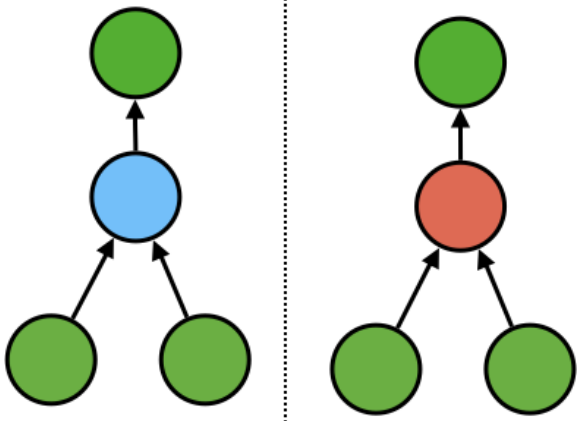
Geometric of ∞-Wasserstein



$\epsilon-$ball in Euclidean space

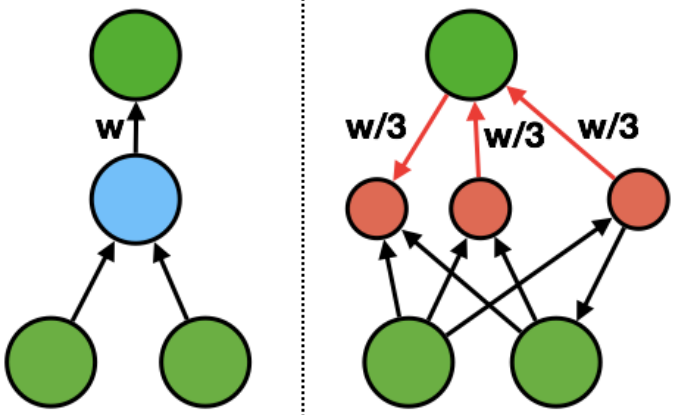


$\epsilon-$ball in ∞-Wasserstein



$[\theta, w] \mapsto [\theta', w]$

Parametric updates

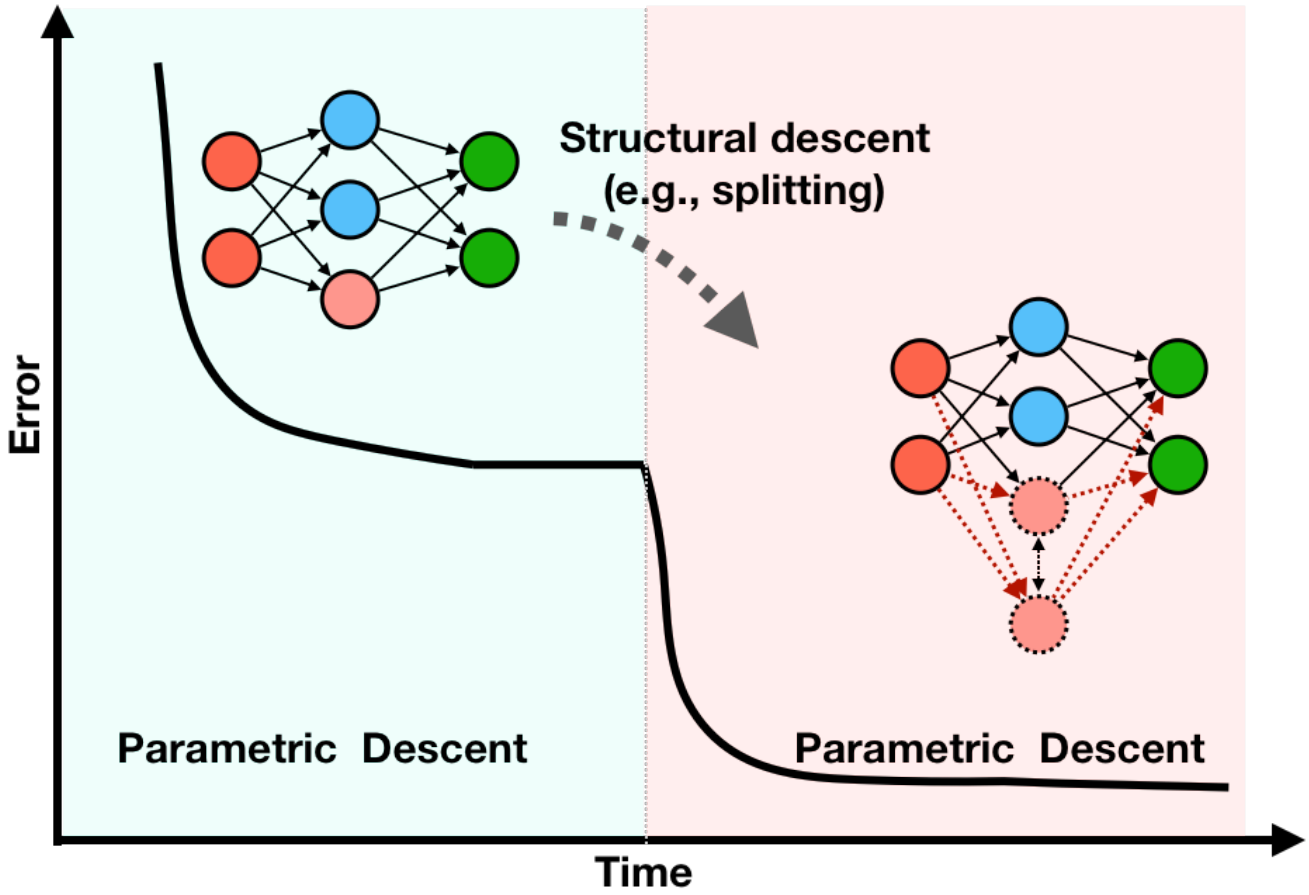


$[\theta, w] \mapsto \{[\theta'_i, w_i]\}, \quad \sum_i w_i = w$

Structural updates (by "splitting" neurons)

∞-Wasserstein Steepest Descent

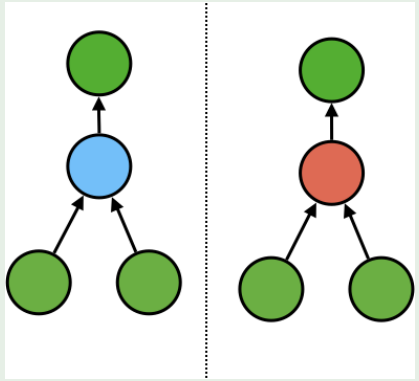
∞-Wasserstein Steepest descent on Θ_∞ alternates between parametric updates and structural updates:



∞-Wasserstein Steepest Descent

- ∞-Wasserstein Steepest descent on Θ_∞ alternates between two phases:

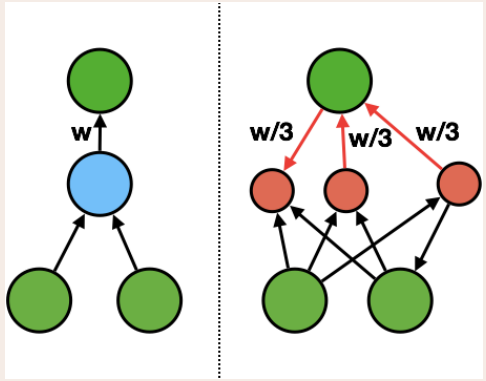
- **Parametric descent within a fixed structure.**
  - Standard gradient descent.
  - Only update parameters; no structural change.
  - Stops when local minima (in Euclidean space) is reached



- **Structural descent to grow the network**
  - Splitting neurons into multiple copies:

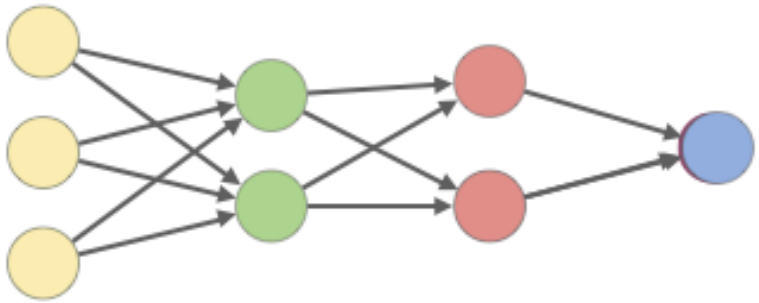
$$\theta, w \mapsto \{\theta_i, w_i\}, \quad \text{with} \quad \sum_i w_i = w.$$

  - Update both parameters and structures.
  - Happens only at parametric stationary points



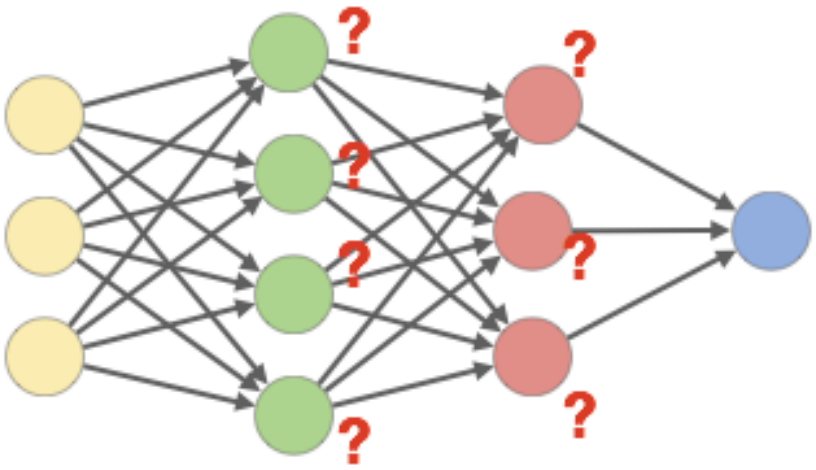
Progressive Training by Splitting Neurons

- Starting from a small net, gradually grow the net during training.
- Grow by "splitting" existing neurons into multiple off-springs.



∞-Wasserstein Steepest Descent = Optimal Splitting

- What neurons should be split first?
- Does splitting decrease the loss? How much?
- How to split a neuron optimally?
- How many copies to split into?



"splitting unstable"

vs.

"splitting stable"

Deriving Optimal Splitting: Simple Case

- Consider the simplest case of a single-neuron network $\sigma(\theta, x)$:

$$L(\theta) = \mathbb{E}_{x\sim\mathcal{D}}[\Phi(\sigma(\theta, x))].$$

- Split it into m neurons $\boldsymbol{\theta} := \{\theta_i\}_{i=1}^m$ with weights $\boldsymbol{w} := \{w_i\}$:

$$Ł(\boldsymbol{\theta}, \boldsymbol{w}) = \mathbb{E}_{x\sim\mathcal{D}}\left[\Phi\left(\sum_{i=1}^m w_i\sigma(\theta_i, x)\right)\right].$$

- Obviously, $Ł(\boldsymbol{\theta}, \boldsymbol{w}) = L(\theta)$ when $\theta_i = \theta$ and $\sum_i w_i = 1$.

- Question: How to choose m and $\{\boldsymbol{\theta},\ \boldsymbol{w}\}$ optimally

$$\min_{m,\boldsymbol{\theta},\boldsymbol{w}} \left\{ Ł(\boldsymbol{\theta}, \boldsymbol{w}) \ \ s.t. \ \ \|\theta_i - \theta\| \le \epsilon, \ \ w_i \ge 0 \ \ , \ \ \sum_{i=1}^m w_i = 1, \ \ \forall i \right\}.$$

- Let $\theta_i = \theta + \epsilon(\delta_{avg} + \delta_i)$, such that $\sum_i w_i \delta_i = 0$.
  - δ_{avg} is the average displacement;
  - δ_i is the splitting direction of θ_i.

- A key decomposition of the augmented loss:

$$Ł(\boldsymbol{\theta},\ \boldsymbol{w}) = \underbrace{L(\theta + \epsilon\delta_{avg})}_{Displacement} + \underbrace{\frac{\epsilon^2}{2} I\!I(\theta, \boldsymbol{\delta}, \boldsymbol{w})}_{Splitting} + O(\epsilon^3)$$

where

$$I\!I(\theta, \boldsymbol{\delta}, \boldsymbol{w}) = \sum_{i=1}^{m} w_i \delta_i^\top S(\theta) \delta_i,$$

and $S(\theta)$ is a "semi-Hessian" matrix called the **"splitting matrix"**:

$$S(\theta) = \mathbb{E}[\Phi'(\sigma(\theta, x)) \nabla^2_{\theta\theta} \sigma(\theta, x)].$$

$$\text{Ł}(\boldsymbol{\theta},\ \boldsymbol{w}) = \underbrace{L(\theta + \epsilon\delta_{avg})}_{\textit{Displacement}} + \underbrace{\frac{\epsilon^2}{2} I\!I(\theta, \boldsymbol{\delta}, \boldsymbol{w})}_{\textit{Splitting}} + O(\epsilon^3)$$

- Optimal splitting:

$$\min_{m,\boldsymbol{\delta},\boldsymbol{w}} \left\{ I\!I(\theta, \boldsymbol{\delta}, \boldsymbol{w}) := \sum_{i=1}^{m} w_i \delta_i^\top S(\theta) \delta_i \right\}$$

  - When $\lambda_{min}(S(\theta)) < 0$, **the optimal strategy is to split the neuron into two copies with equal weights, following the minimum eigen direction**:

$$m = 2, \quad \delta_1 = v_{min}(S(\theta)), \quad \delta_2 = -v_{min}(S(\theta)), \quad w_1 = w_2 = 1/2.$$

  - When $\lambda_{min}(S(\theta)) > 0$, no splitting can decrease the loss. $L(\theta)$ is splitting stable in this case.

- The splitting matrix is a "semi-Hessian" matrix:

$$S(\theta) = \mathbb{E}[\Phi'(\sigma(\theta, x))\nabla^2_{\theta\theta}\sigma(\theta, x)].$$

- Hessian matrix:

$$\nabla^2 L(\theta) = S(\theta) \ + \ T(\theta),$$

  where

$$T(\theta) = \mathbb{E}\left[\Phi''(\sigma(\theta, x))\nabla_\theta \sigma(\theta, x)^{\otimes 2}\right].$$

- $S(\theta)$ is the "easy part" of the Hessian matrix.

More general Case

- For neural networks with n (types of) neurons $\theta^{[1:n]} = \{\theta^{[1]}, \ldots, \theta^{[n]}\}$, splitting each $\theta^{[\ell]}$ into m_ℓ off-springs $\boldsymbol{\theta}^{[\ell]} = \{\theta_i^{[n]}\}_{\ell=1}^{m_\ell}$ with weights $\boldsymbol{w}^{[\ell]} = \{w_i^{[\ell]}\}$,

$$Ł(\boldsymbol{\theta}^{[1:n]}, \boldsymbol{w}^{[1:n]}) = \underbrace{L(\theta^{[1:n]} + \epsilon\delta_{avg}^{[1:n]})}_{displacement} + \frac{\epsilon^2}{2}\sum_{\ell=1}^{n} \underbrace{\mathit{Il}_\ell(\theta^{[1:n]}; \boldsymbol{\delta}^{[\ell]}, \boldsymbol{w}^{[\ell]})}_{splitting} + \mathcal{O}(\epsilon^3).$$

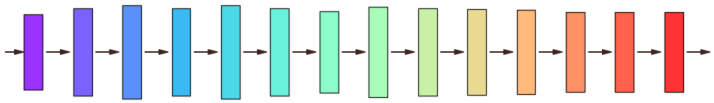
- The overall splitting effect is the sum of individual splittings; there is no crossing term in the splitting matrix, unlike Hessian matrix.
- Neurons of different types can be compared using their splitting matrices.
- Naturally applies to deep neural networks.

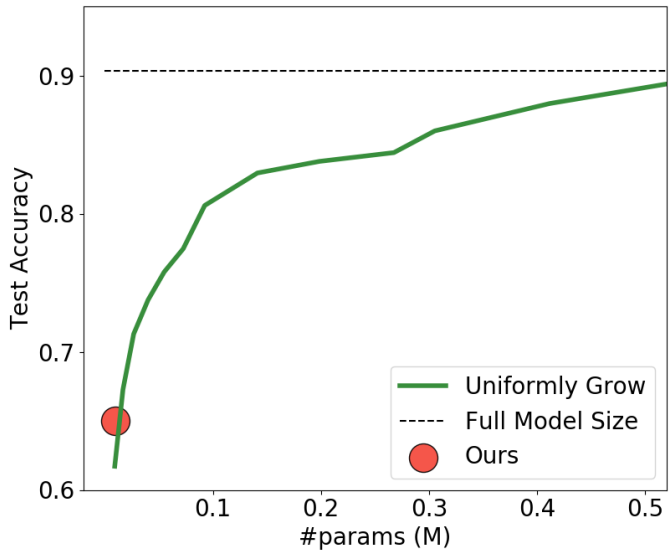
Overall Algorithm

- Repeat:
  - Run standard gradient descent to convergence.
  - Calculate the splitting matrices of all the neurons to be split.
  - Split the neurons with most negative minimum eigenvalues into two copies with equal weights, following the eigenvector directions.

Overall Algorithm

- Repeat:
  - Run standard gradient descent to convergence.
  - Calculate the splitting matrices of all the neurons to be split.
  - Split the neurons with most negative minimum eigenvalues into two copies with equal weights, following the eigenvector directions.





Overall Algorithm

- Repeat:
  - Run standard gradient descent to convergence.
  - Calculate the splitting matrices of all the neurons to be split.
  - Split the neurons with most negative minimum eigenvalues into two copies with equal weights, following the eigenvector directions.

- The computational cost of exact eigen-computation is $\mathcal{O}(nd^3)$.
  - n: the number of neurons;
  - d: the number of parameters of each neuron.

Fast eigen-calculation w/o expanding splitting matrix

- Minimum eigenvalues by gradient descent on Rayleigh quotient:

$$\lambda_{min} = \min_{v} \left\{ R(v) := \frac{v^\top S(\theta) v}{v^\top v} \right\}, \qquad v_{min} = \arg\min_{v} R(v),$$

Gradient descent:

$$v_{t+1} \propto v_t - \epsilon \nabla_v R(v_t), \qquad \nabla_v R(v) \propto S(\theta) v - R(v) v$$

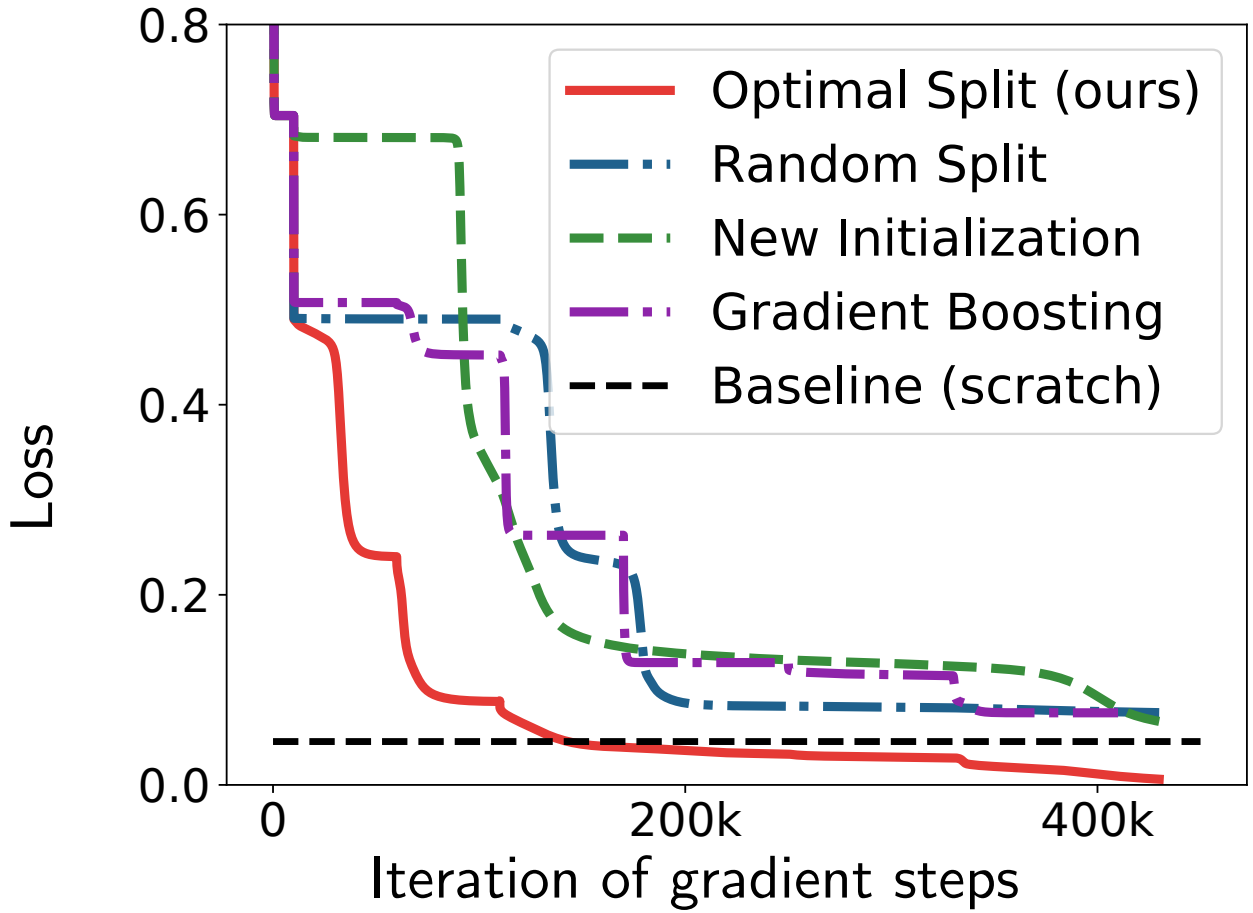
- The matrix-vector product $S(\theta)v$ for $\forall v \in \mathbb{R}^d$ can be calculated with an automatic differentiation trick:

$$S(\theta)v = \nabla_y F(0), \qquad F(y) = \mathbb{E}\left[\Phi(\sigma(\theta, x) + y^\top \nabla^2_{\theta\theta} \sigma(\theta, x) v)\right]$$

Can be done simultaneously for all the neurons.

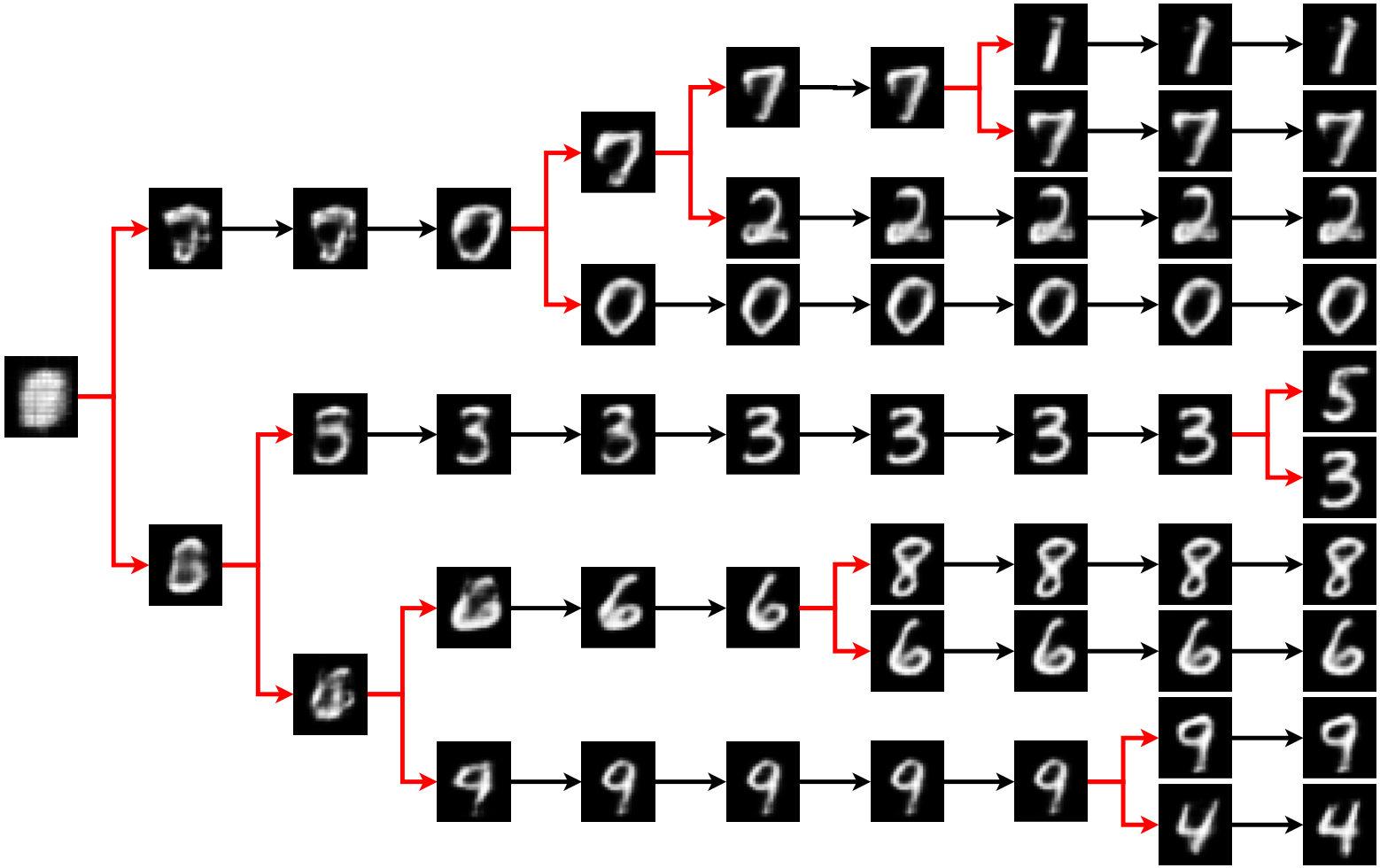
Toy Example

- One-dimensional RBF neural network (with 15 neurons).
- Splitting starting from a single neuron.



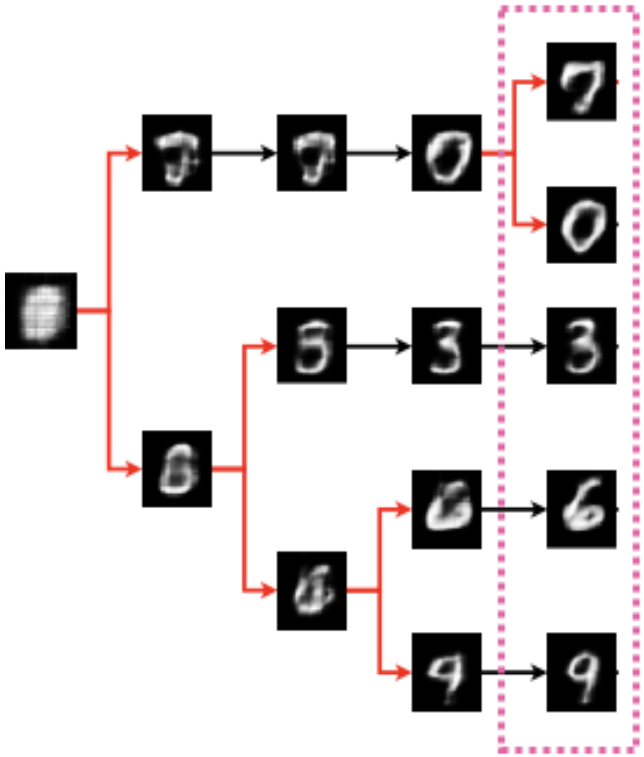
Growing Interpretable Network

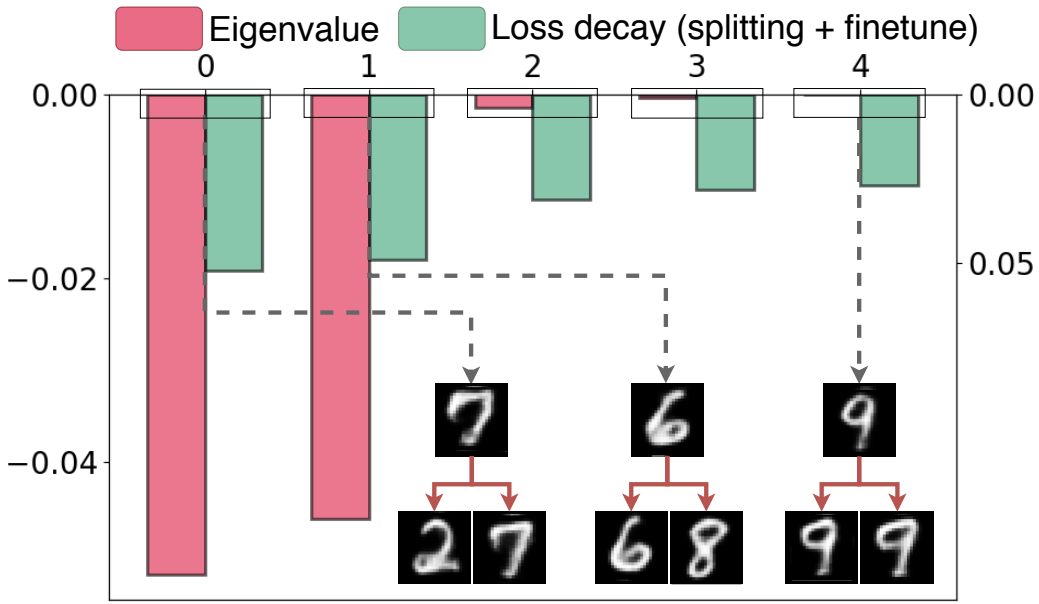
- Training the interpretable neural network by Li et al. 2018[1] .



[1] Li, Oscar et al. "Deep learning for case-based reasoning320 through prototypes: A neural network that explains its predictions". In Thirty-Second AAAI321 Conference on Artificial Intelligence, 2018.

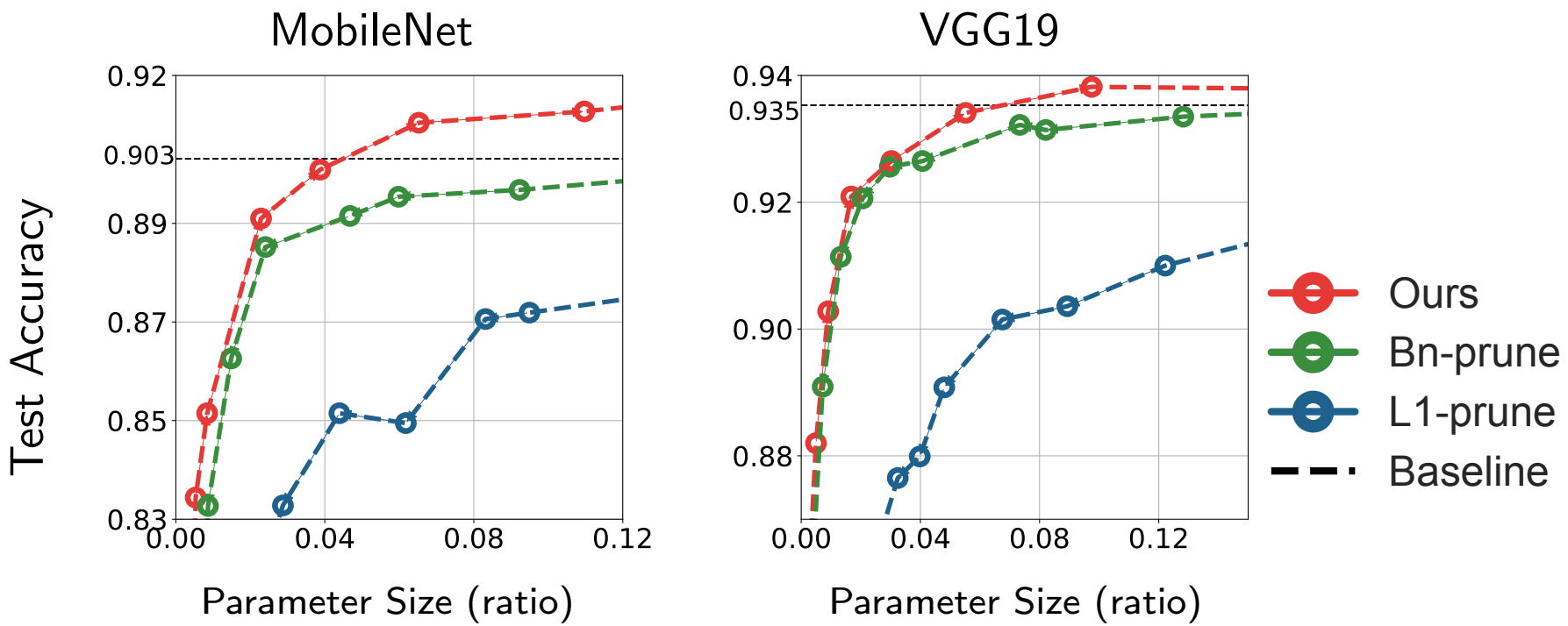
Growing Interpretable Network





Result on CIFAR10

- Compare with pruning methods: batch-normalization-based pruning (Bn-prune) (Liu et al., 2017[2]) and L1-based pruning (L1-prune) (Li et al., 2017[3]).



[2] Liu, Zhuang et al. "Learning efficient convolutional networks through network slimming." In Proceedings of the IEEE International Conference on Computer Vision, pp. 2736?2744, 2017.

[3] Li, Hao et al. "Pruning filters for efficient convnets". International Conference on Learning Representations, 2017.

ImageNet with MobileNetv1 and MobileNetv2

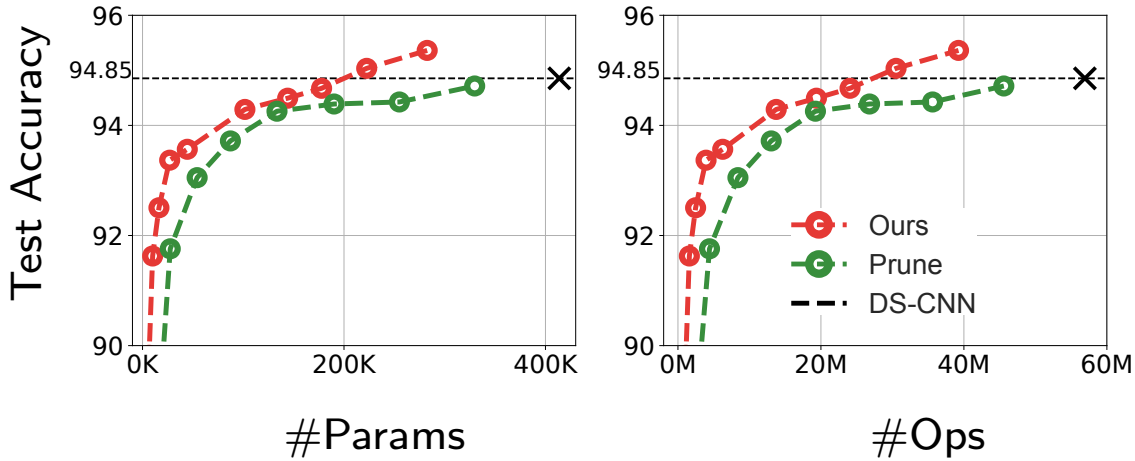
Model	MACs (G)	Top-1 Accuracy	Top-5 Accuracy
MobileNetV1 (1.0x)	0.569	72.93	91.14
Splitting-4	0.561	**73.96**	**91.49**
MobileNetV1 (0.75x)	0.317	70.25	89.49
AMC (He et al., 2018)	0.301	70.50	89.30
Splitting-3	**0.292**	**71.47**	**89.67**
MobileNetV1 (0.5x)	0.150	65.20	86.34
Splitting-2	**0.140**	**68.26**	**87.93**
Splitting-1	0.082	64.06	85.30
Splitting-0 (seed)	0.059	59.20	81.82

Model	MACs (G)	Top-1 Accuracy	Top-5 Accuracy
MobileNetV2 (1.0x)	0.300	72.04	90.57
Splitting-3	**0.298**	**72.84**	**90.83**
MobileNetV2 (0.75x)	0.209	69.80	89.60
AMC (He et al., 2018)	0.210	70.85	89.91
Splitting-2	**0.208**	**71.76**	**90.07**
MobileNetV2 (0.5x)	0.097	65.40	86.40
Splitting-1	**0.095**	**66.53**	**87.00**
Splitting-0 (seed)	0.039	55.61	79.55

Keyword Spotting on Microcontrollers

- Identifying a set of keywords from speech signal
  - e.g., "wake words" for Alexa or Google Assistant.
  - highly resource constrained due to the always-on nature.
  - use benchmark from Zhang et al 2017[4].

Method	Acc	Params (K)	Ops (M)
DNN	86.94	495.7	1.0
CNN	92.64	476.7	25.3
BasicLSTM	93.62	492.6	47.9
LSTM	94.11	495.8	48.4
GRU	94.72	498.0	48.4
CRNN	94.21	485.0	19.3
DS-CNN	94.85	413.7	56.9
Ours	**95.36**	**282.6**	**39.2**



[4] Zhang, Yundong, et al. "Hello edge: Keyword spotting on microcontrollers." arXiv preprint arXiv:1711.07128 (2017).

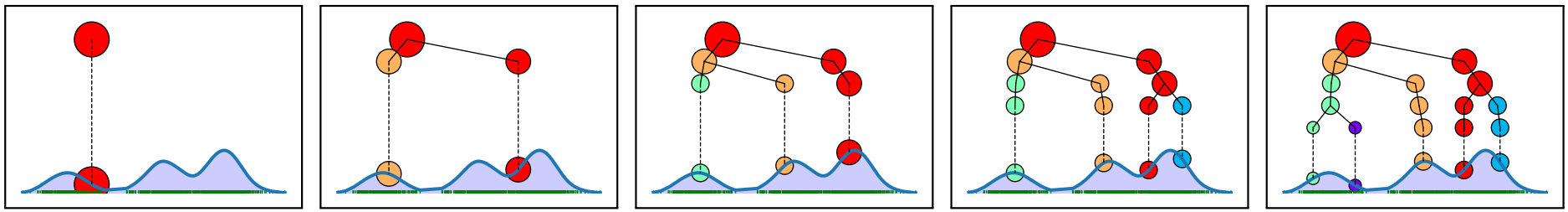
Particle Approximate Inference via Splitting Descent

- Giving a distribution p, find a set of samples $\{\theta_i\}$ to approximate p.
- Can be framed into optimization:

$$\min_{\{\theta_i\}} D(\{\theta_i\}, \quad p),$$

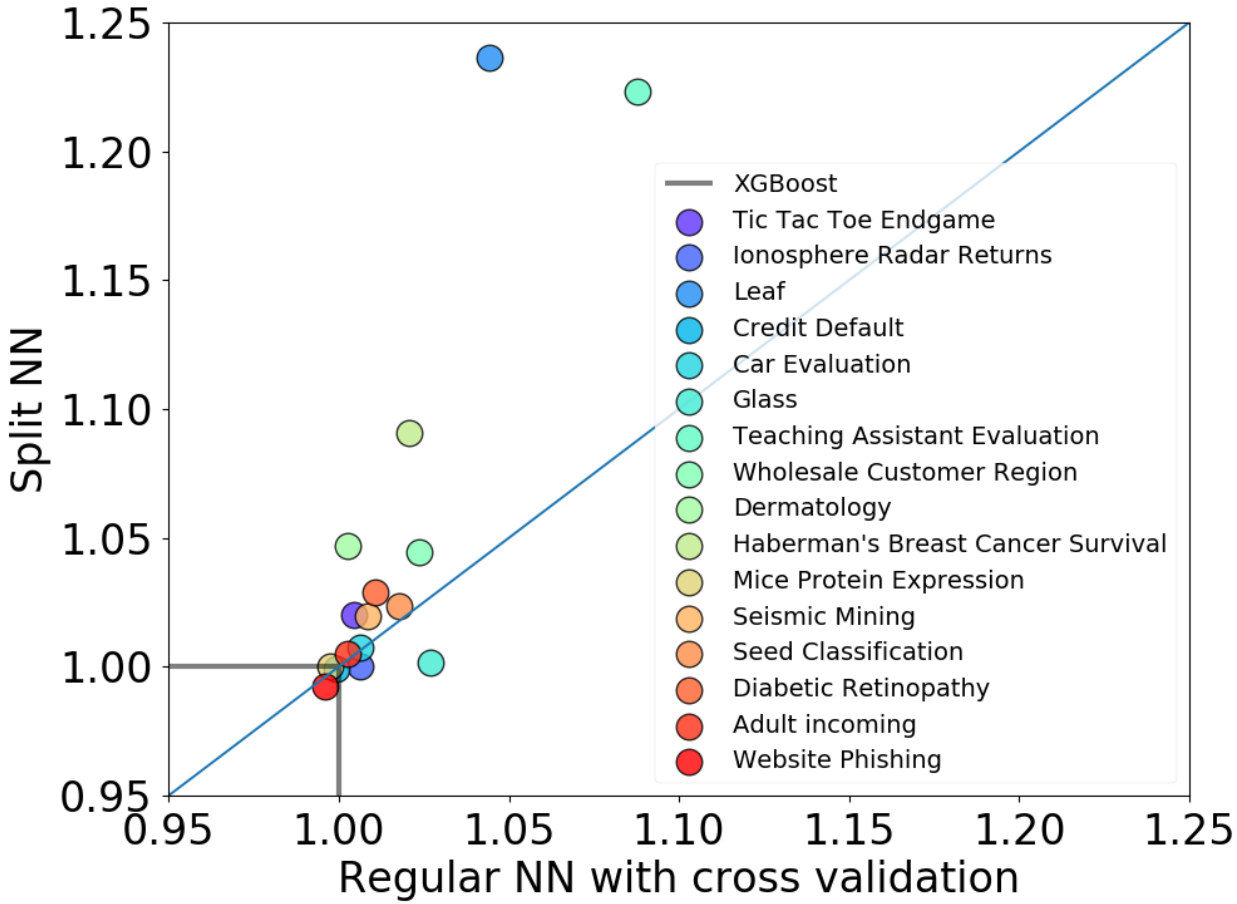
where $D(\cdot, \cdot)$ is a discrepancy measure (e.g., MMD, Stein discrepancy, etc).

- Splitting descent: gradually grow samples by splitting.



Automatic Machine Learning

- Use splitting as a way for automatic neural network structure optimization across different datasets.
- Compared with neural nets with typical cross validation, and xgboost.



Conclusion

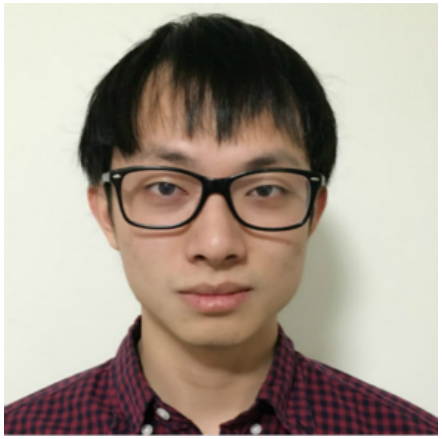
- "Gradient-based" algorithm for neural architecture optimization (NAO).
- Progressive training by splitting neurons.
- Simple and fast, promising in practice.
- Opens a new dimension for energy-efficient NAO.

Reference:

- Liu et al. *Splitting Steepest Descent for Growing Neural Architectures*. NeurIPS 2019
- Wang et al. *Energy-Aware Neural Architecture Optimization with Fast Splitting Steepest Descent*. Arxiv 2019

Thanks!

Dilin Wang



Lemeng Wu

Chengyue Gong