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
  \[ m = [m_1, m_2, \ldots, m_d]. \]

- Let $\Theta_m$ be the space of parameters with structure $m$.

- The Overall model space is
  \[ \Theta_\infty = \bigcup_{m \in \mathbb{N}^d} \Theta_m. \]

- Structure-Parameter Co-optimization:
  \[ \min_{\theta \in \Theta_\infty} L(\theta) \]

- Yields (infinite dimensional) continuous optimization, solve it by (functional) steepest descent!
Recall parametric steepest descent on $\mathbb{R}^d$:

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

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 $\Theta_\infty$

- Equip $\Theta_\infty$ with a proper notion of distance $D(\theta, \theta')$.
- Derive steepest descent on $\Theta_\infty$:

$$\theta_{t+1} \approx \arg \min_{\theta \in \Theta_\infty} \left\{ L(\theta) - L(\theta_t) \quad s.t. \quad D(\theta, \theta_t) \leq \epsilon \right\},$$

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

Key Challenge: How to define distance $D(\cdot, \cdot)$ on $\Theta_\infty$?

- 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:

\[
\begin{align*}
  f(x; \theta) &= \sum_{i=1}^{m} w_i \sigma(\theta_i, x), \\
  f(x; \theta') &= \sum_{i=1}^{m'} w'_i \sigma(\theta'_i, x),
\end{align*}
\]

where \( \theta = \{w_i, \theta_i\}_{i=1}^{m} \) and \( \theta' = \{w'_i, \theta'_i\}_{i=1}^{m'} \) are of different sizes.

∞-Wasserstein metric:

\[
\mathbb{D}_\infty(\theta, \theta') = \inf_{\gamma \in \Gamma} \max_{ij : \gamma_{ij} \neq 0} \|\theta_i - \theta'_j\|
\]

\( \Gamma \): 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 $\infty$-Wasserstein

$\epsilon -$ball in Euclidean space

$\epsilon -$ball in $\infty$-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 $\Theta_\infty$ alternates between parametric updates and structural updates:
\(\infty\)-Wasserstein Steepest Descent

\(\infty\)-Wasserstein Steepest descent on \(\Theta_\infty\) 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.
$\infty$-Wasserstein Steepest Descent $\Rightarrow$ 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”
Consider the simplest case of a single-neuron network $\sigma(\theta, x)$:

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

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

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

Obviously, $L(\theta, w) = L(\theta)$ when $\theta_i = \theta$ and $\sum_i w_i = 1$.

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

$$\min_{m, \theta, w} \left\{ L(\theta, w) \ s.t. \ ||\theta_i - \theta|| \leq \epsilon, \ w_i \geq 0, \ \sum_{i=1}^m w_i = 1, \ \forall i \right\}.$$
Let $\theta_i = \theta + \epsilon(\delta_{avg} + \delta_i)$, such that $\sum_i w_i \delta_i = 0$.

- $\delta_{avg}$ is the average displacement;
- $\delta_i$ is the splitting direction of $\theta_i$.

A key decomposition of the augmented loss:

$$L(\theta, w) = L(\theta + \epsilon \delta_{avg}) + \frac{\epsilon^2}{2} II(\theta, \delta, w) + O(\epsilon^3)$$

where

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

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

$$S(\theta) = \mathbb{E} [\Phi'(\sigma(\theta, x)) \nabla^2_{\theta\theta} \sigma(\theta, x)].$$
\[
L(\theta, \mathbf{w}) = L(\theta + \epsilon \delta_{\text{avg}}) + \frac{\epsilon^2}{2} II(\theta, \delta, \mathbf{w}) + O(\epsilon^3)
\]

- Optimal splitting:

\[
\min_{m, \delta, \mathbf{w}} \left\{ II(\theta, \delta, \mathbf{w}) := \sum_{i=1}^{m} w_i \delta_i^\top S(\theta) \delta_i \right\}
\]

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

\[
m = 2, \quad \delta_1 = \nu_{\text{min}}(S(\theta)), \quad \delta_2 = -\nu_{\text{min}}(S(\theta)), \quad w_1 = w_2 = 1/2.
\]

- When \(\lambda_{\text{min}}(S(\theta)) > 0\), no splitting can decrease the loss. \(L(\theta)\) is splitting stable in this case.
The splitting matrix is a “semi-Hessian” matrix:

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

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 $m_\ell$ off-springs $\theta^{[\ell]} = \{\theta_i^{[n]}\}^{m_\ell}_{\ell=1}$ with weights $w^{[\ell]} = \{w_i^{[\ell]}\}$,

$$L(\theta^{[1:n]}, w^{[1:n]}) = \underbrace{L(\theta^{[1:n]} + \epsilon\delta^{[1:n]}_{\text{avg}})}_{\text{displacement}} + \frac{\epsilon^2}{2} \sum_{\ell=1}^{n} II_{\ell}(\theta^{[1:n]}, \delta^{[\ell]}, w^{[\ell]}) + O(\epsilon^3).$$

- 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.
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 $O(nd^3)$.

- $n$: 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:

\[ \lambda_{min} = \min_v \left\{ R(v) := \frac{v^\top S(\theta) v}{v^\top v} \right\}, \quad v_{min} = \arg \min_v R(v), \]

Gradient descent:

\[ v_{t+1} \propto v_t - \epsilon \nabla_v R(v_t), \quad \nabla_v R(v) \propto S(\theta)v - R(v)v \]

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

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

Can be done simultaneously for all the neurons.
Toy Example

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

![Graph showing the performance of different splitting methods over iterations.](image)
Growing Interpretable Network

- Training the interpretable neural network by Li et al. 2018\(^1\).

---

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\(^3\)).
# ImageNet with MobileNetV1 and MobileNetV2

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Acc</th>
<th>Params (K)</th>
<th>Ops (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN</td>
<td>86.94</td>
<td>495.7</td>
<td>1.0</td>
</tr>
<tr>
<td>CNN</td>
<td>92.64</td>
<td>476.7</td>
<td>25.3</td>
</tr>
<tr>
<td>BasicLSTM</td>
<td>93.62</td>
<td>492.6</td>
<td>47.9</td>
</tr>
<tr>
<td>LSTM</td>
<td>94.11</td>
<td>495.8</td>
<td>48.4</td>
</tr>
<tr>
<td>GRU</td>
<td>94.72</td>
<td>498.0</td>
<td>48.4</td>
</tr>
<tr>
<td>CRNN</td>
<td>94.21</td>
<td>485.0</td>
<td>19.3</td>
</tr>
<tr>
<td>DS-CNN</td>
<td>94.85</td>
<td>413.7</td>
<td>56.9</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td><strong>95.36</strong></td>
<td><strong>282.6</strong></td>
<td><strong>39.2</strong></td>
</tr>
</tbody>
</table>

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:

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

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:

Thanks!

Dilin Wang

Lemeng Wu

Chengyue Gong