DARTS: Differentiable Architecture Search [1]

Recent trends in Automated Machine Learning (AutoML)
(IN2107, IN4954)
Technical University of Munich

Philipp Foth
Munich, 09. June 2021
Quick Review
Quick Review

**NAS with RL** [2]: Search for entire architecture
Quick Review

**NAS with RL** [2]: Search for entire architecture
Quick Review

**NASNet** [3] (and AmoebaNet [4]): Search for cells that get stacked together
Quick Review

**NASNet** [3] (and AmoebaNet [4]): Search for cells that get stacked together
Quick Review

**NASNet** [3] (and AmoebaNet [4]): Search for cells that get stacked together.
Motivation
Motivation

Search cost for CIFAR-10 architecture:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Search method</th>
<th>GPU days</th>
<th>In years</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAS [2]</td>
<td>Reinforcement Learning (RL)</td>
<td>22400</td>
<td>61.3</td>
</tr>
<tr>
<td>AmoebaNet [4]</td>
<td>Evolutionary Algorithm</td>
<td>3150</td>
<td>8.6</td>
</tr>
</tbody>
</table>

Recent trends in AutoML (IN2107, IN4954) | DARTS | Philipp Foth
Motivation

NASNet and AmoebaNet
- Good results
- Inefficient search
Motivation

NASNet and AmoebaNet
● Good results
● Inefficient search

Search space: discrete and non-differentiable $\rightarrow$ RL and Evolution
Motivation

NASNet and AmoebaNet

- Good results
- Inefficient search

**Search space: discrete and non-differentiable → RL and Evolution**

More efficient (faster) search possible with gradient information directly from the search space
Motivation

NASNet and AmoebaNet
- Good results
- Inefficient search

**Search space: discrete** and **non-differentiable** → RL and Evolution

More efficient (faster) search possible with gradient information directly from the search space

**Differentiable** Architecture Search (DARTS)
Search Space: NASNet \cite{3}
Search Space: NASNet\textsuperscript{[3]}

Cell can be represented as a Directed Acyclic Graph
Search Space: NASNet\textsuperscript{[3]}

Cell can be represented as a **Directed Acyclic Graph**
Search Space: NASNet\textsuperscript{[3]}

Cell can be represented as a **Directed Acyclic Graph**

**Nodes** = latent representations  
**Edges** = operations

---

Recent trends in AutoML (IN2107, IN4954) | DARTS | Philipp Foth
Search Space: NASNet\textsuperscript{[3]}

Cell can be represented as a **Directed Acyclic Graph**

- **Nodes** = latent representations
- **Edges** = operations

- 2 input nodes
  (outputs of 2 previous cells)
Search Space: NASNet\textsuperscript{[3]}

Cell can be represented as a Directed Acyclic Graph

Nodes = latent representations

Edges = operations

- 2 input nodes
  (outputs of 2 previous cells)

- 5 intermediate nodes
  (each with 2 edges from previous nodes)
Search Space: NASNet$^3$

Cell can be represented as a **Directed Acyclic Graph**

- **Nodes** = latent representations
- **Edges** = operations

- 2 input nodes (outputs of 2 previous cells)
- 5 intermediate nodes (each with 2 edges from previous nodes)
- 1 output node (concatenate all intermediate nodes)

**Recent trends in AutoML (IN2107, IN4954) | DARTS | Philipp Foth**
Search Space: DARTS\textsuperscript{[1]}
Search Space: DARTS \cite{1}
Search Space: DARTS\textsuperscript{[1]}

Input and output nodes: fixed
Search Space: DARTS\textsuperscript{[1]}

Input and output nodes: fixed

Intermediate nodes: fix to \textcolor{green}{add}
Search Space: DARTS\textsuperscript{[1]}

Input and output nodes: fixed

Intermediate nodes: fix to \textbf{add}

(guarantees that dimension stays the same)
Search Space: DARTS\textsuperscript{[1]}

Input and output nodes: fixed

Intermediate nodes: fix to add

(guarantees that dimension stays the same)

Learning cell = Learning edges

---

Recent trends in AutoML (IN2107, IN4954) | DARTS | Philipp Foth
Search Space: DARTS\textsuperscript{[1]}

Input and output nodes: fixed

Intermediate nodes: fix to \textcolor{red}{add}

(guarantees that dimension stays the same)

\textbf{Learning cell = Learning} \textcolor{red}{edges}

(which operations and which input nodes)
Search Space: Continuous Relaxation
Search Space: Continuous Relaxation

(a) 
(b) 
(c) 
(d)
Search Space: Continuous Relaxation
Search Space: Continuous Relaxation

Mixture of operations through softmax:

\[ \tilde{o}^{(i,j)}(x) = \sum_{o \in 0} \frac{\exp(\alpha^{(i,j)}_o)}{\sum_{o' \in 0} \exp(\alpha^{(i,j)}_{o'})} o(x) \]
Search Space: Continuous Relaxation

Mixture of operations through softmax:

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

$O$ : set of candidate operations
Search Space: Continuous Relaxation

Mixture of operations through softmax:

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

\( \mathcal{O} \): set of candidate operations

\( o(x) \): function applied to latent representation \( x \)
Search Space: Continuous Relaxation

Mixture of operations through softmax:

\[
\bar{o}^{(i,j)}(x) = \frac{\exp(\alpha^{(i,j)}_o)}{\sum_{o' \in \mathcal{O}} \exp(\alpha^{(i,j)}_{o'})} \circ (x)
\]

\(\mathcal{O}\): set of candidate operations

\(\circ(x)\): function applied to latent representation \(x\)

\(\alpha^{(i,j)}\): operation mixing weights for edge \((i, j)\) – “encoding of the architecture”
Search Space: Specific Experiment Settings
Search Space: Specific Experiment Settings

Convolutional cells:
- 7 nodes (2 input, 4 intermediate, 1 output)
- Inputs: outputs of the 2 previous cells (direct and skip connection)
- Operations (O): \{3 \times 3, 5 \times 5\} separable convolutions, \{3 \times 3, 5 \times 5\} dilated separable convolutions, 3 \times 3 max pooling, 3 \times 3 average pooling, identity, zero\}
Search Space: Specific Experiment Settings

Convolutional cells:
- 7 nodes (2 input, 4 intermediate, 1 output)
- Inputs: outputs of the 2 previous cells (direct and skip connection)
- Operations (O): \{3 \times 3, 5 \times 5\} separable convolutions, \{3 \times 3, 5 \times 5\} dilated separable convolutions, 3\times3 max pooling, 3\times3 average pooling, identity, zero\}

Recurrent cells:
- 12 nodes (2 input, 9 intermediate, 1 output)
- Inputs: current input and previous hidden state
- Operations (O): \{linear transformations followed by one of \{tanh, ReLU, sigmoid\} activations, identity, zero\}
Optimization
Optimization

Goal: jointly learn architecture \( \alpha \) and weights \( w \)
Optimization

Goal: jointly learn architecture $\alpha$ and weights $\omega$

Bilevel Optimization:
Optimization

Goal: jointly learn architecture $\alpha$ and weights $\omega$

**Bilevel Optimization:**

*Inner* optimization: find best **weights** on **training** set (with current architecture)
Optimization

Goal: jointly learn architecture $\alpha$ and weights $w$

Bilevel Optimization:

**Inner** optimization: find best **weights** on training set (with current architecture)

**Outer** optimization: find best **architecture** on validation set
Optimization

Goal: jointly learn architecture $\alpha$ and weights $w$

Bilevel Optimization:

**Inner** optimization: find best weights on training set (with current architecture)

**Outer** optimization: find best architecture on validation set

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha)$$

s.t. $w^*(\alpha) = \arg\min_w \mathcal{L}_{train}(w, \alpha)$
Optimization

Goal: jointly learn architecture $\alpha$ and weights $w$

Bilevel Optimization:

**Inner** optimization: find best **weights** on **training** set (with current architecture) - Problematic!

**Outer** optimization: find best **architecture** on **validation** set

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha)$$

s.t. $w^*(\alpha) = \text{argmin}_w \mathcal{L}_{train}(w, \alpha)$
Optimization: Approximation
Optimization: Approximation

Approximate $w^*$
Optimization: Approximation

Approximate $w^*$

First order approximation:

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}(w, \alpha)$$
Optimization: Approximation

Approximate $w^*$

First order approximation:

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}(w, \alpha)$$

Second order approximation (one gradient descent step):

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$$
Optimization: Approximation

With second order approximation still problematic: second order gradient (gradient of gradient)

\[ \nabla_\alpha \mathcal{L}_{val}(\alpha^*, \alpha) \approx \nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha) \]
Optimization: Approximation

With second order approximation still problematic: second order gradient (gradient of gradient)

\[
\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)
\]

With chain rule, can be rewritten as:

\[
\nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi \nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_w \mathcal{L}_{val}(w', \alpha)
\]

where: \( w' = w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha) \)
Optimization: Approximation

Second order gradient leads to very large matrix vector multiplication:

\[ \nabla_{\alpha, w}^2 \mathcal{L}_{\text{train}}(w, \alpha) \nabla_w \mathcal{L}_{\text{val}}(w', \alpha) \text{, where: } w' = w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha) \]
Optimization: Approximation

Second order gradient leads to very large matrix vector multiplication:

$$\nabla_{\alpha,w}^2 \mathcal{L}_{\text{train}}(w, \alpha) \nabla_{w'} \mathcal{L}_{\text{val}}(w', \alpha), \text{ where: } w' = w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha)$$

Can be approximated by finite differences with step size $\epsilon$ (from multivariate Taylor expansion):
Optimization: Approximation

Second order gradient leads to very large matrix vector multiplication:

\[ \nabla^2_{\alpha, w} \mathcal{L}_{\text{train}}(w, \alpha) \nabla_w \mathcal{L}_{\text{val}}(w', \alpha), \text{ where: } w' = w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha) \]

Can be approximated by finite differences with step size \( \epsilon \) (from multivariate Taylor expansion):

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

where \( w^\pm = w \pm \epsilon \nabla_w \mathcal{L}_{\text{val}}(w', \alpha) \)
Optimization: Recap
Optimization: Recap

Cell architecture and network weights optimized together
Optimization: Recap

Cell **architecture** and network **weights** optimized **together**

Only one network is trained during search
Optimization: Recap

Cell architecture and network weights optimized together

Only one network is trained during search

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge $(i, j)$

while not converged do

1. Update architecture $\alpha$ by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$

2. Update weights $w$ by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned $\alpha$. 

Recent trends in AutoML (IN2107, IN4954) | DARTS | Philipp Foth 15
Architecture Discretization
Architecture Discretization

Each node gets assigned the top-k strongest edges = largest $\alpha$’s
Architecture Discretization

Each node gets assigned the top-k strongest edges = largest $\alpha$’s

($k = 2$, only nonzero operations)
Architecture Discretization

Each node gets assigned the top-k strongest edges = largest $\alpha$’s

(k = 2, only nonzero operations)

Resulting discrete cells:
Architecture Discretization

Each node gets assigned the top-k strongest edges = largest $\alpha$’s

(k = 2, only nonzero operations)

Resulting discrete cells:

(get retrained, do not keep the $w$’s)
Results
Results

- Random Search strong baseline
- Bilevel Optimization is essential
Results

- Random Search strong baseline
- Bilevel Optimization is essential

<table>
<thead>
<tr>
<th>Architecture searched on CIFAR-10</th>
<th>CIFAR-10 Test Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Search</td>
<td>3.29 ± 0.15</td>
</tr>
<tr>
<td>DARTS (Coordinate descent on all data)</td>
<td>4.16 ± 0.16</td>
</tr>
<tr>
<td>DARTS (Gradient descent on all data)</td>
<td>3.56 ± 0.10</td>
</tr>
<tr>
<td>DARTS (bilevel optimization, first order approximation)</td>
<td>3.00 ± 0.14</td>
</tr>
<tr>
<td>DARTS (bilevel optimization, second order approximation)</td>
<td>2.76 ± 0.09</td>
</tr>
</tbody>
</table>
Results: CIFAR-10
## Results: CIFAR-10

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Test Error (%)</th>
<th>Params (M)</th>
<th>Search cost (GPU days)</th>
<th>Search method</th>
</tr>
</thead>
<tbody>
<tr>
<td>DenseNet-BC</td>
<td>3.46</td>
<td>25.6</td>
<td>-</td>
<td>manual</td>
</tr>
<tr>
<td>NASNet-A + cutout</td>
<td>2.65</td>
<td>3.3</td>
<td>2000</td>
<td>RL</td>
</tr>
<tr>
<td>AmoebaNet-B + cutout</td>
<td>2.55 ± 0.05</td>
<td>2.8</td>
<td>3150</td>
<td>evolution</td>
</tr>
<tr>
<td>DARTS (second order) + cutout</td>
<td>2.76 ± 0.09</td>
<td>3.3</td>
<td>4</td>
<td>gradient-based</td>
</tr>
</tbody>
</table>

(DARTS repeated 4 times with different initializations, best one selected)
Results

Convolutional cells (searched on CIFAR-10)
- Also transferable to ImageNet
- Competitive with NASNet
Results

Convolutional cells (searched on CIFAR-10)
- Also transferable to ImageNet
- Competitive with NASNet

Recurrent cells (searched on PTB)
- State-of-the-art results on PBT
- Less transferrable to WT2
Conclusion
Conclusion

Advantages:
Conclusion

**Advantages:**
Much more efficient architecture search, can be performed without massive resources
Conclusion

Advantages:
Much more efficient architecture search, can be performed without massive resources
You can search for architectures for your own projects: DARTS GitHub
Conclusion

Advantages:
Much more efficient architecture search, can be performed without massive resources
You can search for architectures for your own projects: DARTS GitHub

Potential issues:
Conclusion

Advantages:
Much more efficient architecture search, can be performed without massive resources
You can search for architectures for your own projects: DARTS GitHub

Potential issues:
Mismatch between optimized mixture cell and discretized version
Conclusion

Advantages:
Much more efficient architecture search, can be performed without massive resources
  You can search for architectures for your own projects: DARTS GitHub

Potential issues:
Mismatch between optimized mixture cell and discretized version
  Only mentioned by authors, no quantification given
Conclusion - further work in NAS
Conclusion - further work in NAS

DARTS direction:
Conclusion - further work in NAS

DARTS direction:

Made NAS much more accessible, which lead to a lot of follow up work
Conclusion - further work in NAS

DARTS direction:

Made NAS much more accessible, which lead to a lot of follow up work

- **P-DARTS** [7], **FairDARTS** [8], **DARTS+** [9], **sharpDARTS** [10] (better performance)
- **PC-DARTS** [11] (reduce computational cost, use larger batch size, better performance)
- **UnNAS** [12] (unsupervised NAS, without human annotated labels)
- **ProxylessNAS** [13] (reduce computational cost, search on target dataset, low latency objective, better performance)
- And many, many more…
Conclusion - further work in NAS

RL and evolution direction:
Conclusion - further work in NAS

RL and evolution direction:

*MnasNet* [5] $\rightarrow$ multi-objective optimization: maximize accuracy and minimize FLOPS
Conclusion - further work in NAS

RL and evolution direction:


Was used for *EfficientNet* [6]
Questions?
References