DARTS: Differentiable ARchiTecture Search

Source Code, arXiv

M. K. Ipsit
ipsit.ml@gmail.com
https://ipsitmantri.github.io/

Department of Electrical Engineering
Indian Institute of Technology Bombay

Paper Info

- Hanxiao Liu, Karen Simonyan, Yiming Yang
- Conference paper at ICLR 2019
- Neural Architecture Search (NAS)

Overview

- Scalable Architecture Search

Overview

- Scalable Architecture Search
- Efficient search in a differential space using gradient descent

Overview

- Scalable Architecture Search
- Efficient search in a differential space using gradient descent
- Faster than existing NAS algorithms

Overview

- Scalable Architecture Search
- Efficient search in a differential space using gradient descent
- Faster than existing NAS algorithms
- Unified framework - works for both CNN and RNN architectures

Overview

- Scalable Architecture Search
- Efficient search in a differential space using gradient descent
- Faster than existing NAS algorithms
- Unified framework - works for both CNN and RNN architectures
- **Continuous Relaxation** of architectural search

Introduction

Introduction

Motivation

- Coming up with the right architecture for a given dataset is a manual and a difficult task
- Q: Is it possible to do this algorithmically and automate it?

Introduction

Motivation

- Coming up with the right architecture for a given dataset is a manual and a difficult task
- Q: Is it possible to do this algorithmically and automate it?

Prior Attempts

- Highly competitive performance on image classification, object detection
- Computationally demanding :(- 2000 GPU days for CIFAR-10

Introduction

Motivation

- Coming up with the right architecture for a given dataset is a manual and a difficult task
- Q: Is it possible to do this algorithmically and automate it?

Prior Attempts

- Highly competitive performance on image classification, object detection
- Computationally demanding :(- 2000 GPU days for CIFAR-10

Dominant Approaches

- Reinforcement Learning, Evolution, Monte Carlo Tree Search, Sequential Model-Based Optimization, Bayesian Optimization

Introduction

Motivation

- Coming up with the right architecture for a given dataset is a manual and a difficult task
- Q: Is it possible to do this algorithmically and automate it?

Prior Attempts

- Highly competitive performance on image classification, object detection
- Computationally demanding :(- 2000 GPU days for CIFAR-10

Dominant Approaches

- Reinforcement Learning, Evolution, Monte Carlo Tree Search, Sequential Model-Based Optimization, Bayesian Optimization

Proposed Speedup Methods

- Imposing a particular structure on the search space
- Weights/performance prediction for each individual architecture
- Weight sharing/inheritance across multiple architectures

Basic Idea

- Search for the optimal architecture in a search space

Basic Idea

- Search for the optimal architecture in a search space
- Search space is often discrete

Basic Idea

- Search for the optimal architecture in a search space
- Search space is often discrete
- NAS: A black-box optimization problem over a discrete domain

Basic Idea

- Search for the optimal architecture in a search space
- Search space is often discrete
- NAS: A black-box optimization problem over a discrete domain
  - Requires a large number of architecture evaluations
- Scalability is a fundamental challenge!

Paper Highlights

- Search Space Relaxation – to be continuous

Paper Highlights

- Search Space Relaxation – to be continuous
- Architecture is optimized with respect to its validation set performance via **gradient descent** – **bilevel optimization**.

Paper Highlights

- Search Space Relaxation – to be continuous
- Architecture is optimized with respect to its validation set performance via **gradient descent** – **bilevel optimization**.
- Competitive performance with S.O.T.A using orders of magnitude less computation resources

Paper Highlights

- Search Space Relaxation – to be continuous
- Architecture is optimized with respect to its validation set performance via **gradient descent** – **bilevel optimization**.
- Competitive performance with S.O.T.A using orders of magnitude less computation resources
- Outperforms ENAS

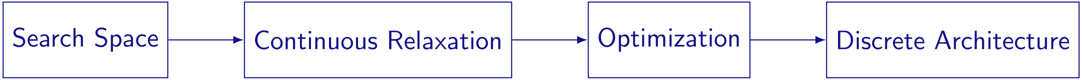
Paper Highlights

- Search Space Relaxation – to be continuous
- Architecture is optimized with respect to its validation set performance via **gradient descent** – **bilevel optimization**.
- Competitive performance with S.O.T.A using orders of magnitude less computation resources
- Outperforms ENAS
- Doesn't involve controllers, hypernetworks or performance predictors

Paper Highlights

- Search Space Relaxation – to be continuous
- Architecture is optimized with respect to its validation set performance via **gradient descent** – **bilevel optimization**.
- Competitive performance with S.O.T.A using orders of magnitude less computation resources
- Outperforms ENAS
- Doesn't involve controllers, hypernetworks or performance predictors
- Simple, yet generic enough to handle convolutional and recurrent architectures.

Process Flow



Search Space

- Final Architecture is made up of different computation cells

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG
  - Ordered sequence of N nodes $\{x^{(1)}, \ldots, x^{(N)}\}$

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG
  - Ordered sequence of N nodes $\{x^{(1)}, \ldots, x^{(N)}\}$
  - Each $x^{(i)}$ is a latent representation

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG
  - Ordered sequence of N nodes $\{x^{(1)}, \dots, x^{(N)}\}$
  - Each $x^{(i)}$ is a latent representation
  - Each directed edge (i, j) is associated with some operation $o^{(i,j)}$ which transforms $x^{(i)}$ into $x^{(j)}$.

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG
  - Ordered sequence of N nodes $\{x^{(1)}, \dots, x^{(N)}\}$
  - Each $x^{(i)}$ is a latent representation
  - Each directed edge (i, j) is associated with some operation $o^{(i,j)}$ which transforms $x^{(i)}$ into $x^{(j)}$.
  -

$$x^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right) \tag{1}$$

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG
  - Ordered sequence of N nodes $\{x^{(1)}, \ldots, x^{(N)}\}$
  - Each $x^{(i)}$ is a latent representation
  - Each directed edge (i, j) is associated with some operation $o^{(i,j)}$ which transforms $x^{(i)}$ into $x^{(j)}$.
  - $$x^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right) \quad (1)$$
  - `zero` operation – to indicate lack of an edge

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG
  - Ordered sequence of N nodes $\{x^{(1)}, \ldots, x^{(N)}\}$
  - Each $x^{(i)}$ is a latent representation
  - Each directed edge (i, j) is associated with some operation $o^{(i,j)}$ which transforms $x^{(i)}$ into $x^{(j)}$.
  -

$$x^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right) \tag{1}$$

  - `zero` operation – to indicate lack of an edge
- Each cell has 2 inputs and one output.

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG
  - Ordered sequence of N nodes $\{x^{(1)}, \ldots, x^{(N)}\}$
  - Each $x^{(i)}$ is a latent representation
  - Each directed edge (i, j) is associated with some operation $o^{(i,j)}$ which transforms $x^{(i)}$ into $x^{(j)}$.
  - $$x^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right) \tag{1}$$
  - `zero` operation – to indicate lack of an edge
- Each cell has 2 inputs and one output.
  - One inputs is the output of previous cell in case of conv.

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG
  - Ordered sequence of N nodes $\{x^{(1)}, \ldots, x^{(N)}\}$
  - Each $x^{(i)}$ is a latent representation
  - Each directed edge (i, j) is associated with some operation $o^{(i,j)}$ which transforms $x^{(i)}$ into $x^{(j)}$.
  - $$x^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right) \tag{1}$$
  - `zero` operation – to indicate lack of an edge
- Each cell has 2 inputs and one output.
  - One inputs is the output of previous cell in case of conv.
  - One input is hidden state of previous cell in case of recurr.

Search Space

- Final Architecture is made up of different computation cells
  - Stacking them gives convolutional architectures
  - Connecting them gives recurrent architectures
- Each computation cell is a DAG
  - Ordered sequence of N nodes $\{x^{(1)}, \ldots, x^{(N)}\}$
  - Each $x^{(i)}$ is a latent representation
  - Each directed edge (i, j) is associated with some operation $o^{(i,j)}$ which transforms $x^{(i)}$ into $x^{(j)}$.
  - $$x^{(j)} = \sum_{i<j} o^{(i,j)}\left(x^{(i)}\right) \tag{1}$$
  - `zero` operation – to indicate lack of an edge
- Each cell has 2 inputs and one output.
  - One inputs is the output of previous cell in case of conv.
  - One input is hidden state of previous cell in case of recurr.
  - Output is obtained by a reduction operation on all intermediate nodes.

Continuous Relaxation

Continuous Relaxation

- $\mathcal{O}$ – set of candidate operations (e.g. conv, relu, maxpool, zero)

Continuous Relaxation

- $\mathcal{O}$ – set of candidate operations (e.g. conv, relu, maxpool, zero)
- Relaxing categorical choice of a particular op. to a softmax weighted average over $\mathcal{O}$

Continuous Relaxation

- $\mathcal{O}$ – set of candidate operations (e.g. conv, relu, maxpool, zero)
- Relaxing categorical choice of a particular op. to a softmax weighted average over $\mathcal{O}$

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x) \tag{2}$$

Continuous Relaxation

- $\mathcal{O}$ – set of candidate operations (e.g. conv, relu, maxpool, zero)
- Relaxing categorical choice of a particular op. to a softmax weighted average over $\mathcal{O}$

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x) \tag{2}$$

- $\alpha^{(i,j)} = \{\alpha_o^{(i,j)}\} \in \mathbb{R}^{|\mathcal{O}|}$ are op. mixing weights, which are continuous.

Continuous Relaxation

- $\mathcal{O}$ – set of candidate operations (e.g. conv, relu, maxpool, zero)
- Relaxing categorical choice of a particular op. to a softmax weighted average over $\mathcal{O}$

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x) \tag{2}$$

- $\alpha^{(i,j)} = \{\alpha_o^{(i,j)}\} \in \mathbb{R}^{|\mathcal{O}|}$ are op. mixing weights, which are continuous.
- Continuous to discrete: replace with most likely op.

Optimization

Optimization

- **Architecture Search – Learning a set of continuous variables $\alpha^{(i,j)}$**

Optimization

- **Architecture Search – Learning a set of continuous variables $\alpha^{(i,j)}$**
- Optimize validation loss using gradient descent for optimal mixing weights

Optimization

- **Architecture Search – Learning a set of continuous variables** $\alpha^{(i,j)}$
- Optimize validation loss using gradient descent for optimal mixing weights
- Jointly learn α and weights w within all mixed ops.

Optimization

- **Architecture Search – Learning a set of continuous variables $\alpha^{(i,j)}$**
- Optimize validation loss using gradient descent for optimal mixing weights
- Jointly learn α and weights w within all mixed ops.
- If $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ respectively denote the train and validation loss, then our problem is to find optimal (w^*, α^*)

Optimization

- **Architecture Search – Learning a set of continuous variables $\alpha^{(i,j)}$**
- Optimize validation loss using gradient descent for optimal mixing weights
- Jointly learn α and weights w within all mixed ops.
- If $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ respectively denote the train and validation loss, then our problem is to find optimal (w^*, α^*)

$$\min_{\alpha} \mathcal{L}_{val}(w^*, \alpha) \text{ where } w^* = \arg\min_{w} \mathcal{L}_{train}(w, \alpha^*) \tag{3}$$

Optimization

- **Architecture Search – Learning a set of continuous variables $\alpha^{(i,j)}$**
- Optimize validation loss using gradient descent for optimal mixing weights
- Jointly learn α and weights w within all mixed ops.
- If $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ respectively denote the train and validation loss, then our problem is to find optimal (w^*, α^*)

$$\min_{\alpha} \mathcal{L}_{val}(w^*, \alpha) \text{ where } w^* = \arg\min_{w} \mathcal{L}_{train}(w, \alpha^*) \tag{3}$$

- This is a **bilevel optimization problem**

Optimization

- **Architecture Search – Learning a set of continuous variables $\alpha^{(i,j)}$**
- Optimize validation loss using gradient descent for optimal mixing weights
- Jointly learn α and weights w within all mixed ops.
- If $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ respectively denote the train and validation loss, then our problem is to find optimal (w^*, α^*)

$$\min_{\alpha} \mathcal{L}_{val}(w^*, \alpha) \text{ where } w^* = \arg\min_{w} \mathcal{L}_{train}(w, \alpha^*) \tag{3}$$

- This is a **bilevel optimization problem**
  - Often arises in the gradient-based hyperparameter optimization

Optimization

- **Architecture Search – Learning a set of continuous variables $\alpha^{(i,j)}$**
- Optimize validation loss using gradient descent for optimal mixing weights
- Jointly learn α and weights w within all mixed ops.
- If $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ respectively denote the train and validation loss, then our problem is to find optimal (w^*, α^*)

$$\min_{\alpha} \mathcal{L}_{val}(w^*, \alpha) \text{ where } w^* = \arg\min_{w} \mathcal{L}_{train}(w, \alpha^*) \tag{3}$$

- This is a **bilevel optimization problem**
  - Often arises in the gradient-based hyperparameter optimization
  - Here α is the upper level and w is the lower level

Optimization

- **Architecture Search – Learning a set of continuous variables $\alpha^{(i,j)}$**
- Optimize validation loss using gradient descent for optimal mixing weights
- Jointly learn α and weights w within all mixed ops.
- If $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ respectively denote the train and validation loss, then our problem is to find optimal (w^*, α^*)

$$\min_{\alpha} \mathcal{L}_{val}(w^*, \alpha) \text{ where } w^* = \arg\min_{w} \mathcal{L}_{train}(w, \alpha^*) \tag{3}$$

- This is a **bilevel optimization problem**
  - Often arises in the gradient-based hyperparameter optimization
  - Here α is the upper level and w is the lower level
  - Equivalent formulation

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) \text{ s.t. } w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}(w, \alpha) \tag{4}$$

Optimization

- **Architecture Search – Learning a set of continuous variables $\alpha^{(i,j)}$**
- Optimize validation loss using gradient descent for optimal mixing weights
- Jointly learn α and weights w within all mixed ops.
- If $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ respectively denote the train and validation loss, then our problem is to find optimal (w^*, α^*)

$$\min_{\alpha} \mathcal{L}_{val}(w^*, \alpha) \text{ where } w^* = \arg\min_{w} \mathcal{L}_{train}(w, \alpha^*) \tag{3}$$

- This is a **bilevel optimization problem**
  - Often arises in the gradient-based hyperparameter optimization
  - Here α is the upper level and w is the lower level
  - Equivalent formulation

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) \text{ s.t. } w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}(w, \alpha) \tag{4}$$

- We use gradient descent

Speeding up GD

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}\left(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha\right) \tag{5}$$

Speeding up GD

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}\left(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha\right) \tag{5}$$

- Intuitive to Nesterov momentum update

Speeding up GD

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}\Big(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha\Big) \tag{5}$$

- Intuitive to Nesterov momentum update
- First GD on w, and then on α to speed up.

Speeding up GD

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}\left(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha\right) \tag{5}$$

- Intuitive to Nesterov momentum update
- First GD on w, and then on α to speed up.
- Using chain rule, we have

Speeding up GD

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}\left(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha\right) \tag{5}$$

- Intuitive to Nesterov momentum update
- First GD on w, and then on α to speed up.
- Using chain rule, we have

$$\nabla_\alpha \mathcal{L}_{val}\left(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha\right) = \nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi \nabla^2_{\alpha, w} \mathcal{L}_{train}(w, \alpha) \cdot \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \tag{6}$$

Speeding up GD

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}\left(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha\right) \tag{5}$$

- Intuitive to Nesterov momentum update
- First GD on w, and then on α to speed up.
- Using chain rule, we have

$$\nabla_\alpha \mathcal{L}_{val}\left(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha\right) = \nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi \nabla^2_{\alpha, w} \mathcal{L}_{train}(w, \alpha) \cdot \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \tag{6}$$

- Notice the expensive computation i.e., product of Hessian and gradient – $\mathbf{O}(|\alpha||\mathbf{w}|)$

Further Speedup

Further Speedup

- Using finite difference approximation for hessian-gradient product in 6

Further Speedup

- Using finite difference approximation for hessian-gradient product in 6
- Define for an $\epsilon > 0$, $w^{\pm} = w \pm \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$. Then

Further Speedup

- Using finite difference approximation for hessian-gradient product in 6
- Define for an $\epsilon > 0$, $w^{\pm} = w \pm \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$. Then

$$\nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \cdot \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon} \tag{7}$$

Further Speedup

- Using finite difference approximation for hessian-gradient product in 6
- Define for an $\epsilon > 0$, $w^{\pm} = w \pm \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$. Then

$$\nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \cdot \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_{\alpha} \mathcal{L}_{train}(w^+, \alpha) - \nabla_{\alpha} \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon} \tag{7}$$

- This is $\mathbf{O}(|\alpha| + |\mathbf{w}|)$

Further Speedup

- Using finite difference approximation for hessian-gradient product in 6
- Define for an $\epsilon > 0$, $w^{\pm} = w \pm \epsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$. Then

$$\nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \cdot \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon} \tag{7}$$

- This is $\mathbf{O}(|\alpha| + |\mathbf{w}|)$
- **First Order Approximation**: Take $\xi = 0$. Leads to a faster heuristic but worse performance
- **Second Order Approximation**: The case when $\xi \neq 0$

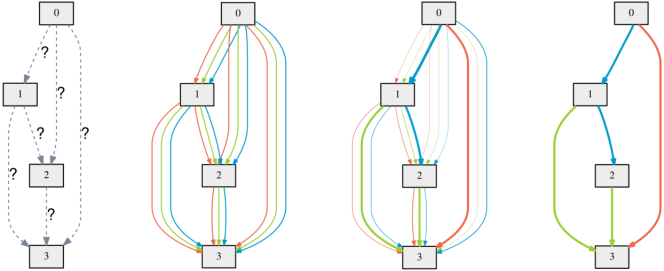
Deriving Discrete Architectures

Deriving Discrete Architectures

- Retain top-k strongest operations to form each node (excluding zero ops).
- For comparison with existing works, $k = 2$ for conv. and $k = 1$ for recurr.
- Zero ops are excluded as they do not affect the final classification outcome due to the presence of batch norm.

Deriving Discrete Architectures

- Retain top-k strongest operations to form each node (excluding zero ops).
- For comparison with existing works, $k = 2$ for conv. and $k = 1$ for recurr.
- Zero ops are excluded as they do not affect the final classification outcome due to the presence of batch norm.



(a) (b) (c) (d)

Combining Everything

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 α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$
   ($\xi = 0$ if using first-order approximation)
2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned α.

Experiments & Results

- Datasets: CIFAR-10 and Penn TreeBank (PTB)

Table 1: Comparison with state-of-the-art image classifiers on CIFAR-10 (lower error rate is better). Note the search cost for DARTS does not include the selection cost (1 GPU day) or the final evaluation cost by training the selected architecture from scratch (1.5 GPU days).

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	#ops	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	–	manual
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	2000	13	RL
NASNet-A + cutout (Zoph et al., 2018)†	2.83	3.1	2000	13	RL
BlockQNN (Zhong et al., 2018)	3.54	39.8	96	8	RL
AmoebaNet-A (Real et al., 2018)	3.34 ± 0.06	3.2	3150	19	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	19	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	19	evolution
Hierarchical evolution (Liu et al., 2018b)	3.75 ± 0.12	15.7	300	6	evolution
PNAS (Liu et al., 2018a)	3.41 ± 0.09	3.2	225	8	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	6	RL
ENAS + cutout (Pham et al., 2018b)*	2.91	4.2	4	6	RL
Random search baseline‡ + cutout	3.29 ± 0.15	3.2	4	7	random
DARTS (first order) + cutout	3.00 ± 0.14	3.3	1.5	7	gradient-based
DARTS (second order) + cutout	2.76 ± 0.09	3.3	4	7	gradient-based

* Obtained by repeating ENAS for 8 times using the code publicly released by the authors. The cell for final evaluation is chosen according to the same selection protocol as for DARTS.

† Obtained by training the corresponding architectures using our setup.

‡ Best architecture among 24 samples according to the validation error after 100 training epochs.

Experiments & Results

Table 2: Comparison with state-of-the-art language models on PTB (lower perplexity is better). Note the search cost for DARTS does not include the selection cost (1 GPU day) or the final evaluation cost by training the selected architecture from scratch (3 GPU days).

Architecture	Perplexity valid	Perplexity test	Params (M)	Search Cost (GPU days)	#ops	Search Method
Variational RHN (Zilly et al., 2016)	67.9	65.4	23	–	–	manual
LSTM (Merity et al., 2018)	60.7	58.8	24	–	–	manual
LSTM + skip connections (Melis et al., 2018)	60.9	58.3	24	–	–	manual
LSTM + 15 softmax experts (Yang et al., 2018)	58.1	56.0	22	–	–	manual
NAS (Zoph & Le, 2017)	–	64.0	25	1e4 CPU days	4	RL
ENAS (Pham et al., 2018b)*	68.3	63.1	24	0.5	4	RL
ENAS (Pham et al., 2018b)†	60.8	58.6	24	0.5	4	RL
Random search baseline‡	61.8	59.4	23	2	4	random
DARTS (first order)	60.2	57.6	23	0.5	4	gradient-based
DARTS (second order)	58.1	55.7	23	1	4	gradient-based

* Obtained using the code (Pham et al., 2018a) publicly released by the authors.

† Obtained by training the corresponding architecture using our setup.

‡ Best architecture among 8 samples according to the validation perplexity after 300 training epochs.

Experiments & Results

Table 3: Comparison with state-of-the-art image classifiers on ImageNet in the mobile setting.

Architecture	Test Error (%) top-1	Test Error (%) top-5	Params (M)	$+\times$ (M)	Search Cost (GPU days)	Search Method
Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	1448	–	manual
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	–	manual
ShuffleNet $2\times$ ($g = 3$) (Zhang et al., 2017)	26.3	–	$\sim$5	524	–	manual
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	564	2000	RL
NASNet-B (Zoph et al., 2018)	27.2	8.7	5.3	488	2000	RL
NASNet-C (Zoph et al., 2018)	27.5	9.0	4.9	558	2000	RL
AmoebaNet-A (Real et al., 2018)	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B (Real et al., 2018)	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C (Real et al., 2018)	24.3	7.6	6.4	570	3150	evolution
PNAS (Liu et al., 2018a)	25.8	8.1	5.1	588	$\sim$225	SMBO
DARTS (searched on CIFAR-10)	26.7	8.7	4.7	574	4	gradient-based

Conclusion

Conclusion

- DARTS is a simple yet efficient architecture search algorithm for both convolutional and recurrent networks

Conclusion

- DARTS is a simple yet efficient architecture search algorithm for both convolutional and recurrent networks
- Outperforms non-differentiable architecture search methods on image classification and language modeling tasks with very high efficiency improvement

Conclusion

- DARTS is a simple yet efficient architecture search algorithm for both convolutional and recurrent networks
- Outperforms non-differentiable architecture search methods on image classification and language modeling tasks with very high efficiency improvement

Improvements

Conclusion

- DARTS is a simple yet efficient architecture search algorithm for both convolutional and recurrent networks
- Outperforms non-differentiable architecture search methods on image classification and language modeling tasks with very high efficiency improvement

Improvements

- Perturbation based selection method. See DARTS-PT

Conclusion

- DARTS is a simple yet efficient architecture search algorithm for both convolutional and recurrent networks
- Outperforms non-differentiable architecture search methods on image classification and language modeling tasks with very high efficiency improvement

Improvements

- Perturbation based selection method. See DARTS-PT
- Operations with largest architecture parameters are selected if we use top-k selection – Can be avoided

Conclusion

- DARTS is a simple yet efficient architecture search algorithm for both convolutional and recurrent networks
- Outperforms non-differentiable architecture search methods on image classification and language modeling tasks with very high efficiency improvement

Improvements

- Perturbation based selection method. See DARTS-PT
- Operations with largest architecture parameters are selected if we use top-k selection – Can be avoided
- Performance-aware architecture derivation schemes

References

- H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX
- R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh, "Rethinking Architecture Selection in Differentiable NAS," 2021. [Online]. Available: https://openreview.net/forum?id=PKubaeJkw3
- H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient Neural Architecture Search via Parameter Sharing," CoRR, vol. abs/1802.03268, 2018, [Online]. Available: http://arxiv.org/abs/1802.03268
- B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing Neural Network Architectures using Reinforcement Learning," CoRR, vol. abs/1611.02167, 2016, [Online]. Available: http://arxiv.org/abs/1611.02167
- A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-Shot Model Architecture Search through HyperNetworks," CoRR, vol. abs/1708.05344, 2017, [Online]. Available: http://arxiv.org/abs/1708.05344

References

- Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, Kevin Murphy; Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 19-34
- B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," CoRR, vol. abs/1611.01578, 2016, [Online]. Available: http://arxiv.org/abs/1611.01578
- E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search", AAAI, vol. 33, no. 01, pp. 4780-4789, Jul. 2019.
- R. Negrinho and G. Gordon, "Deeparchitect: Automatically designing and training deep architectures," arXiv preprint arXiv:1704.08792, 2017.
- K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," Advances in neural information processing systems, vol. 31, 2018.