DARTS: Differentiable Architecture Search [1]

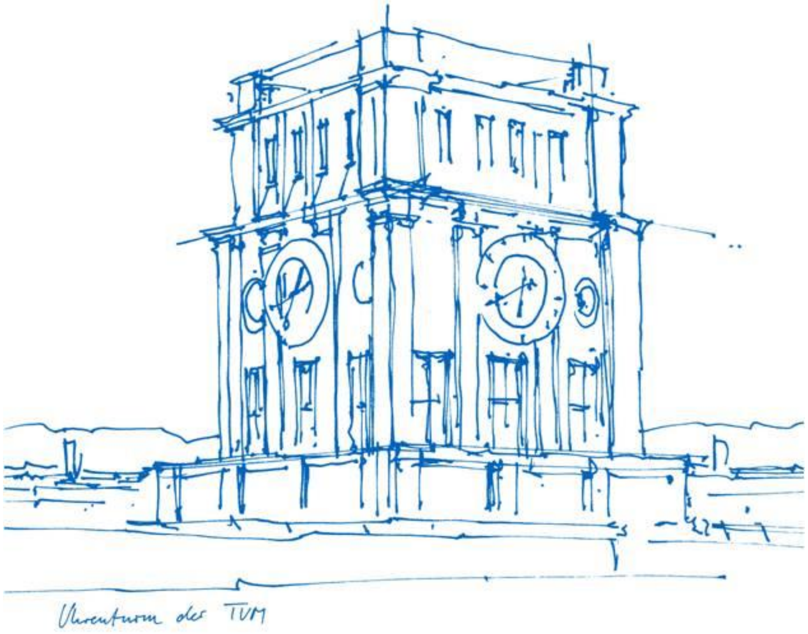
Recent trends in Automated Machine Learning (AutoML)

(IN2107, IN4954)

Technical University of Munich

Philipp Foth

Munich, 09. June 2021



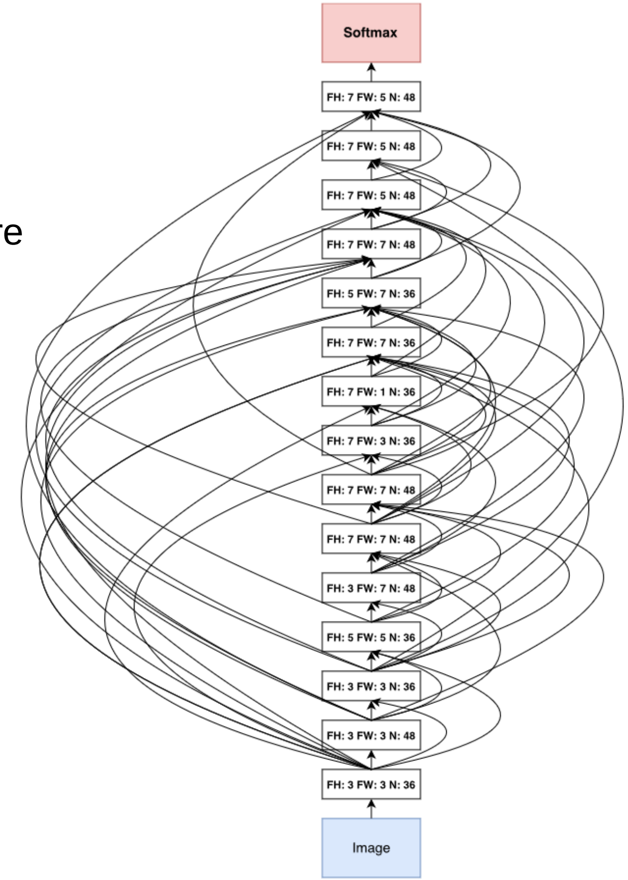
Quick Review

Quick Review

NAS with RL [2]: Search for entire architecture

Quick Review

NAS with RL [2]: Search for entire architecture

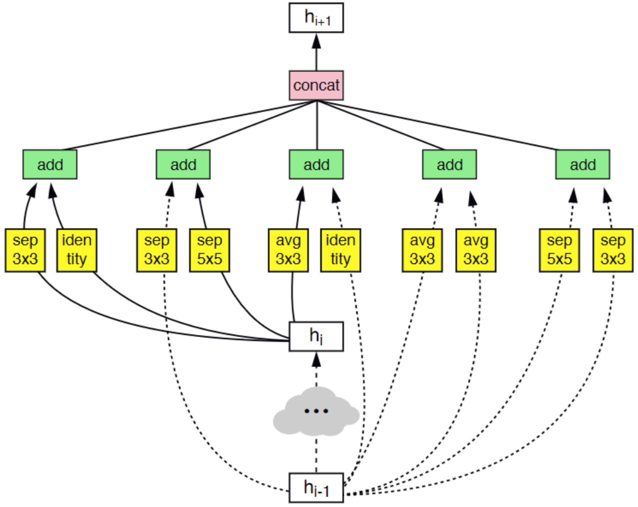


Quick Review

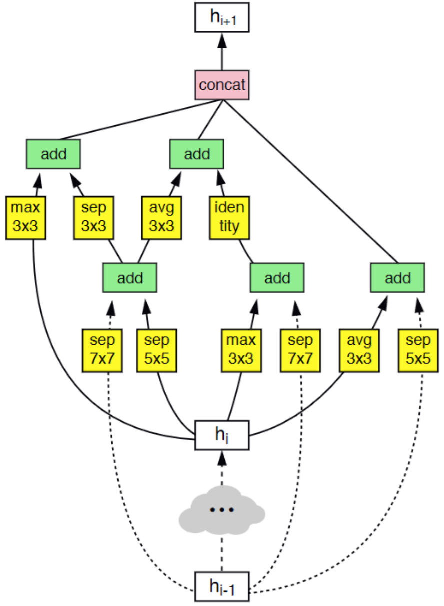
NASNet [3] (and AmoebaNet [4]): Search for cells that get stacked together

Quick Review

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



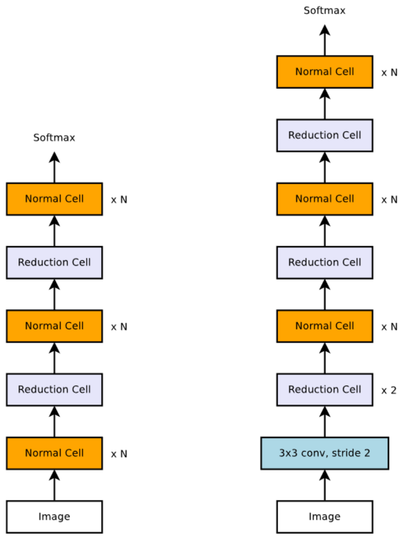
Normal Cell



Reduction Cell

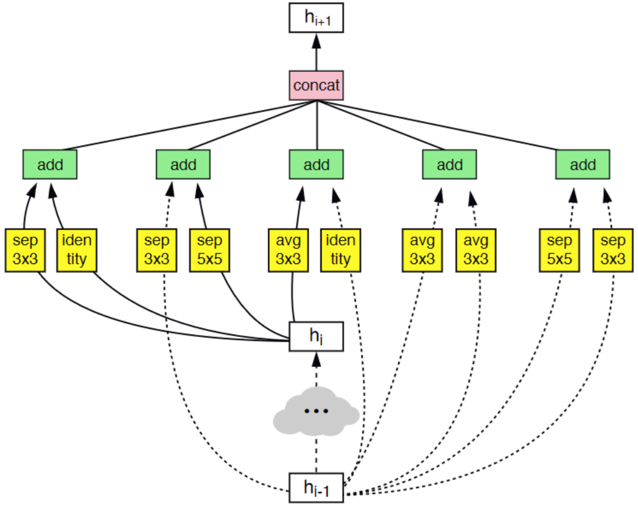
Quick Review

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

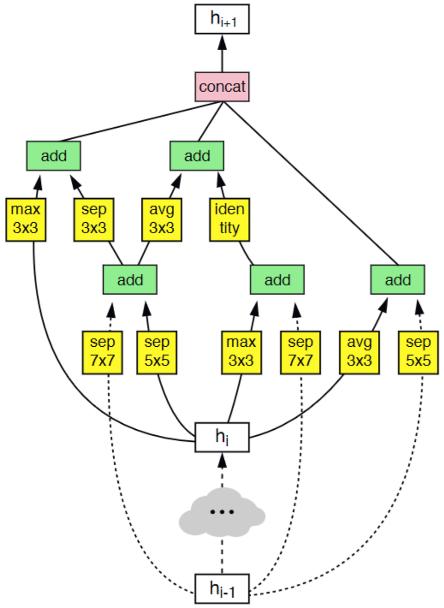


CIFAR10 Architecture

ImageNet Architecture



Normal Cell



Reduction Cell

Motivation

Motivation

Search cost for CIFAR-10 architecture:

Architecture	Search method	GPU days	In years
NAS [2]	Reinforcement Learning (RL)	22400	61.3
NASNet [3]	RL	2000	5.5
AmoebaNet [4]	Evolutionary Algorithm	3150	8.6

Motivation

NASNet and AmoebaNet

- Good results
- Inefficient search

Motivation

NASNet and AmoebaNet

- Good results
- Inefficient search

Search space: discrete and **non-differentiable** → 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)

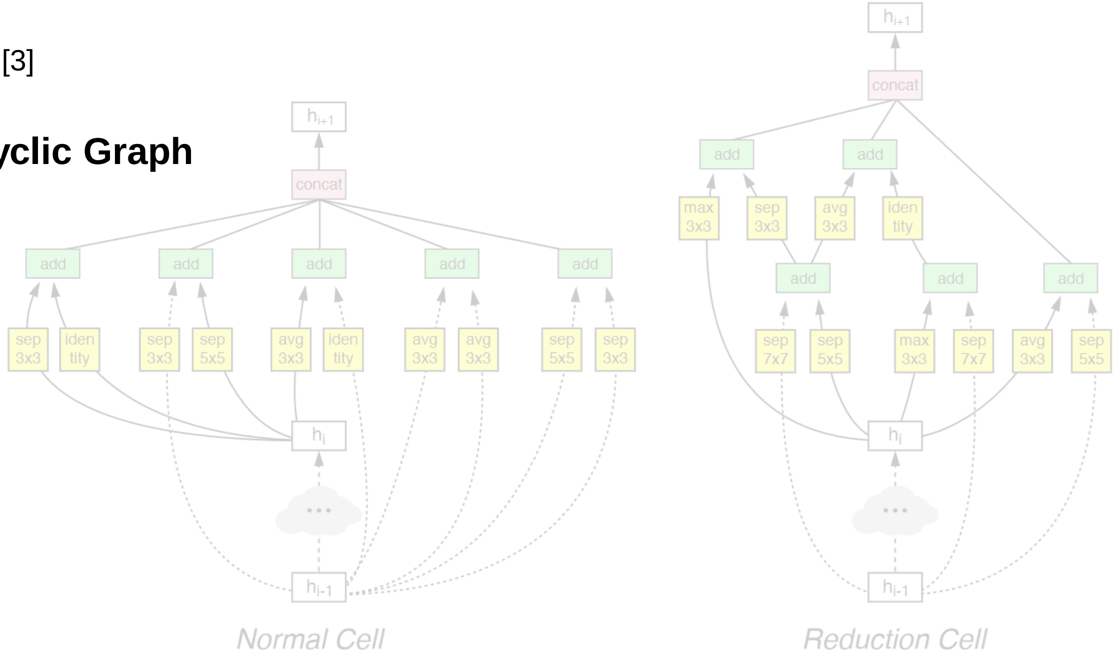
Search Space: NASNet [3]

Search Space: NASNet [3]

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

Search Space: NASNet [3]

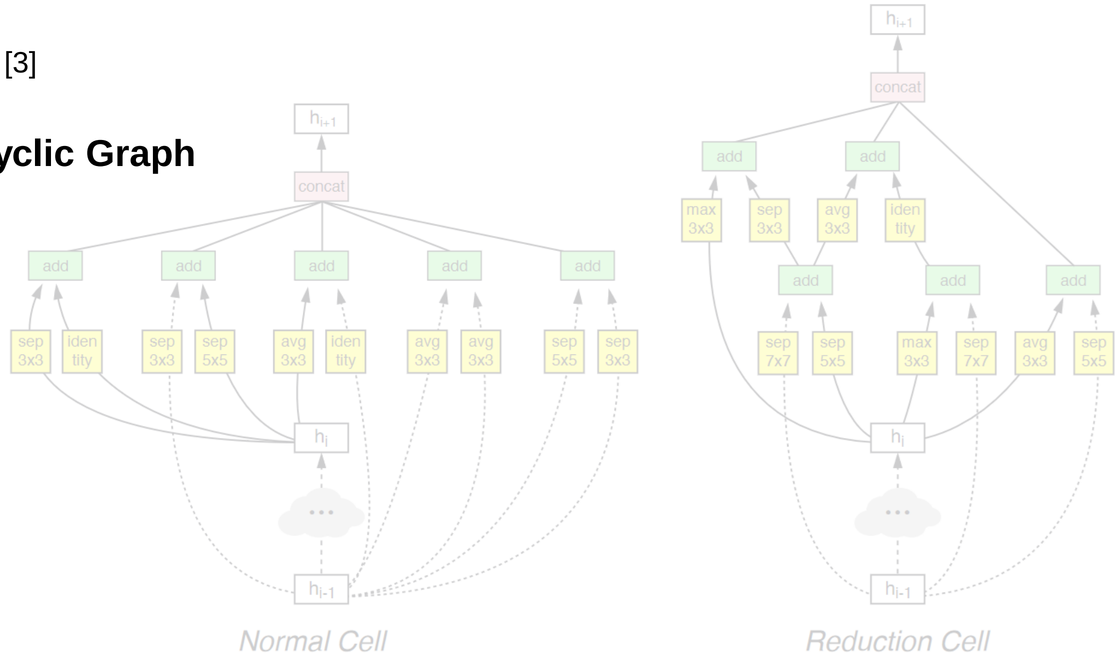
Cell can be represented as a **Directed Acyclic Graph**



Search Space: NASNet [3]

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

Nodes = latent representations
Edges = operations

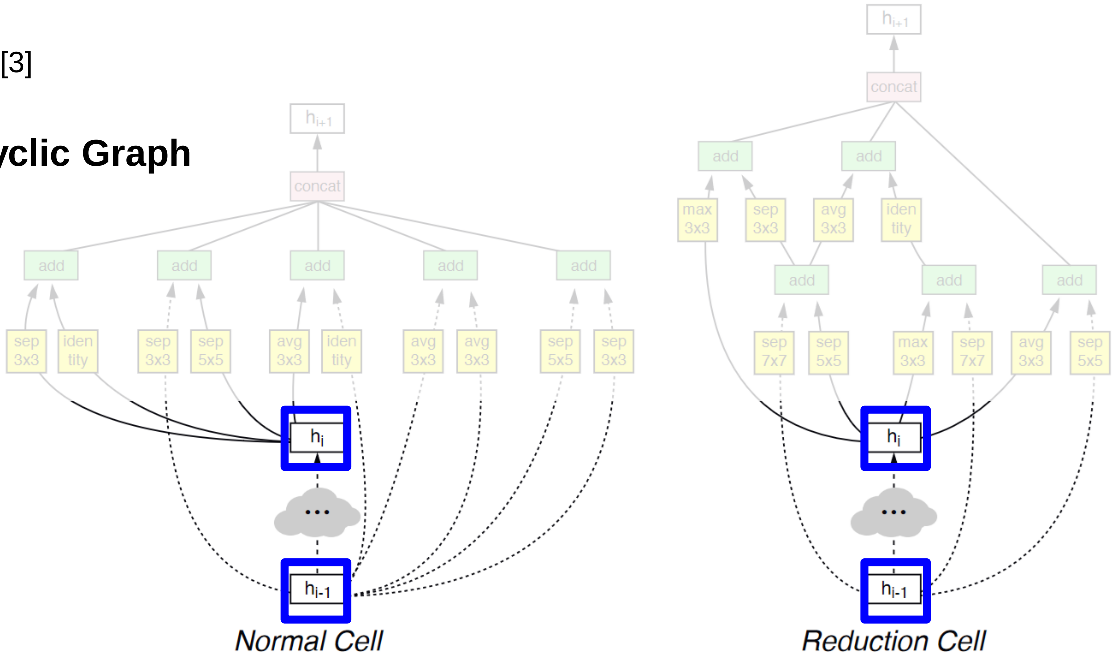


Search Space: NASNet [3]

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

Nodes = latent representations
Edges = operations

- 2 input nodes
  (outputs of 2 previous cells)



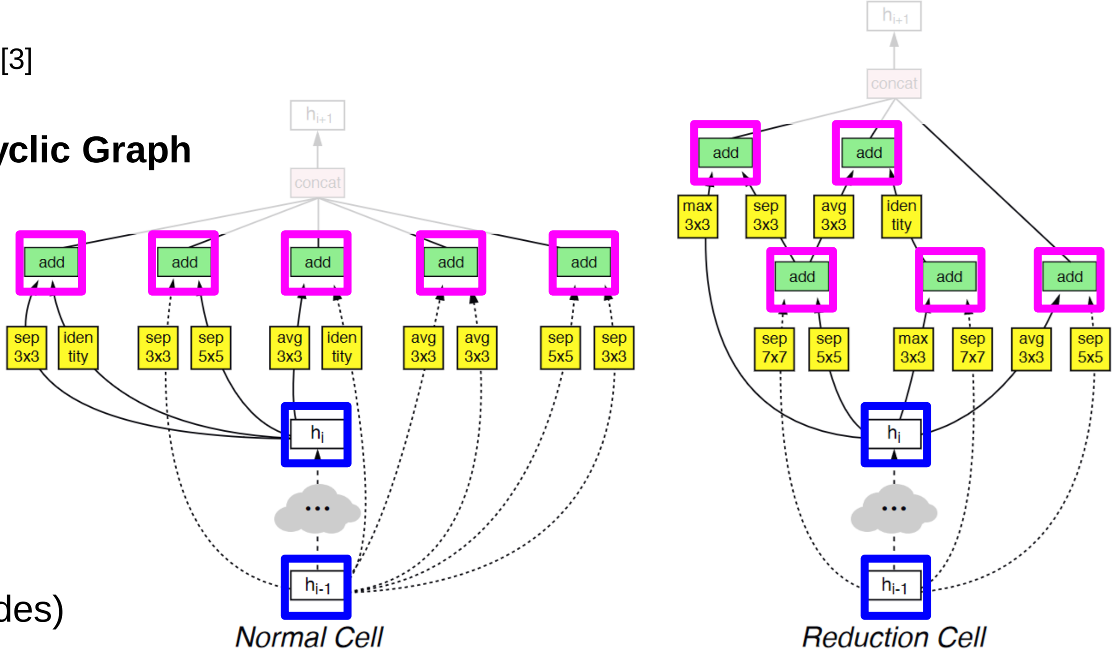
Search Space: NASNet [3]

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)

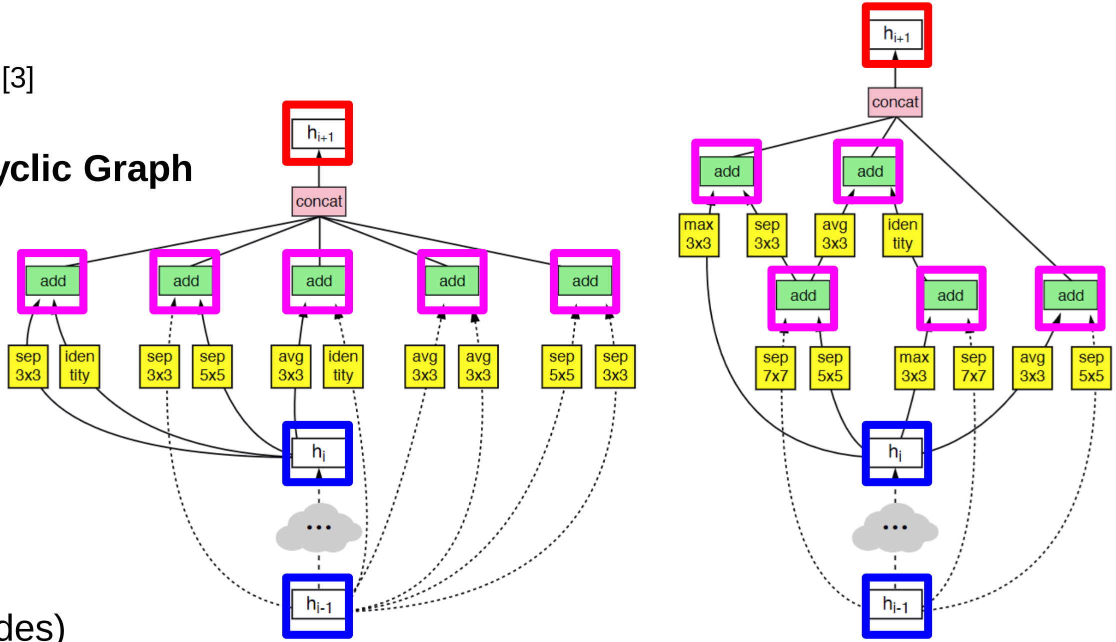


Search Space: NASNet [3]

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

Nodes = latent representations
Edges = operations

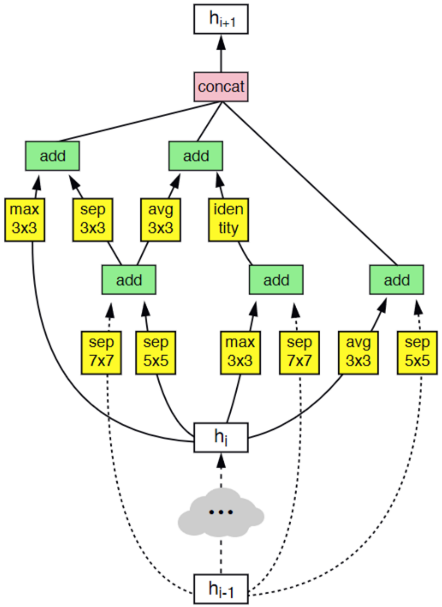
- 2 input nodes
  (outputs of 2 previous cells)
- 5 intermediate nodes
  (each with 2 edges from previous nodes)
- 1 output node (concatenate all intermediate nodes)



Normal Cell *Reduction Cell*

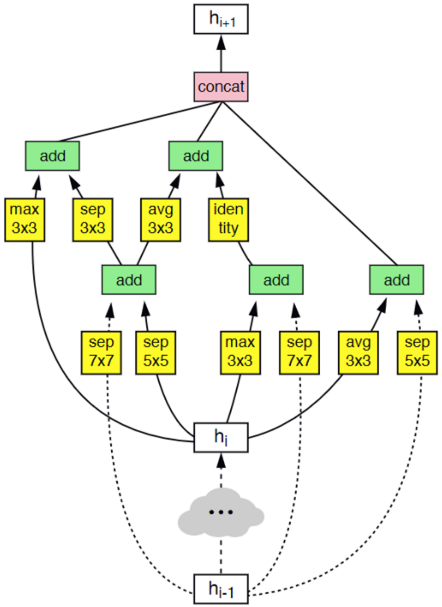
Search Space: DARTS [1]

Search Space: DARTS [1]



Search Space: DARTS [1]

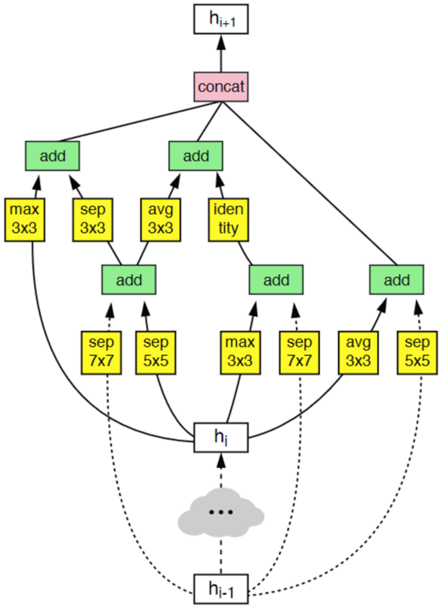
Input and output nodes: fixed



Search Space: DARTS [1]

Input and output nodes: fixed

Intermediate nodes: fix to add

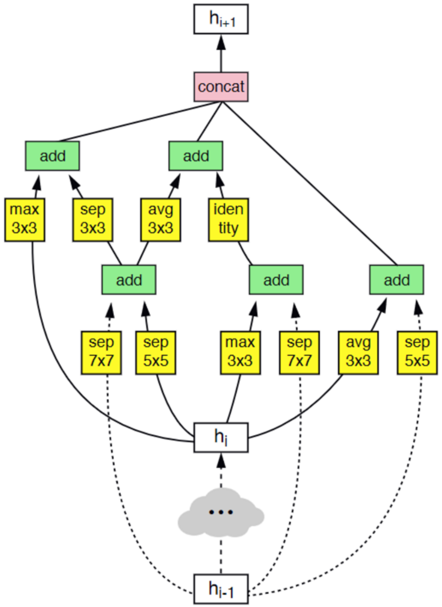


Search Space: DARTS [1]

Input and output nodes: fixed

Intermediate nodes: fix to add

(guarantees that dimension stays the same)



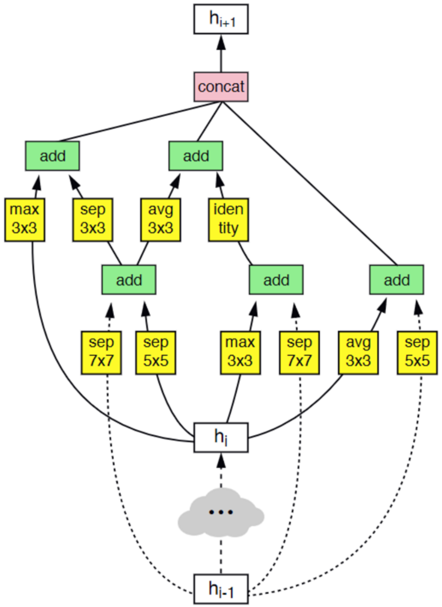
Search Space: DARTS [1]

Input and output nodes: fixed

Intermediate nodes: fix to add

(guarantees that dimension stays the same)

Learning cell = Learning edges



Search Space: DARTS [1]

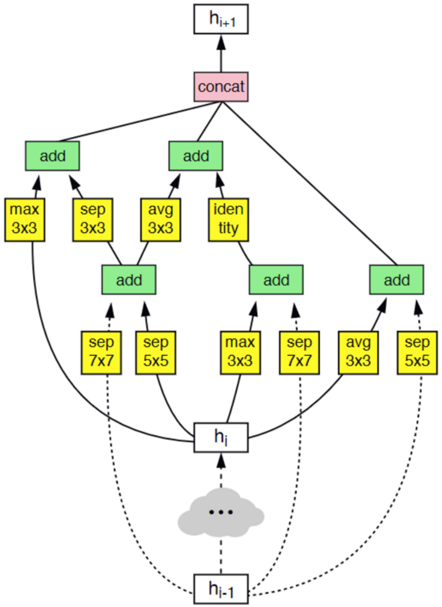
Input and output nodes: fixed

Intermediate nodes: fix to add

(guarantees that dimension stays the same)

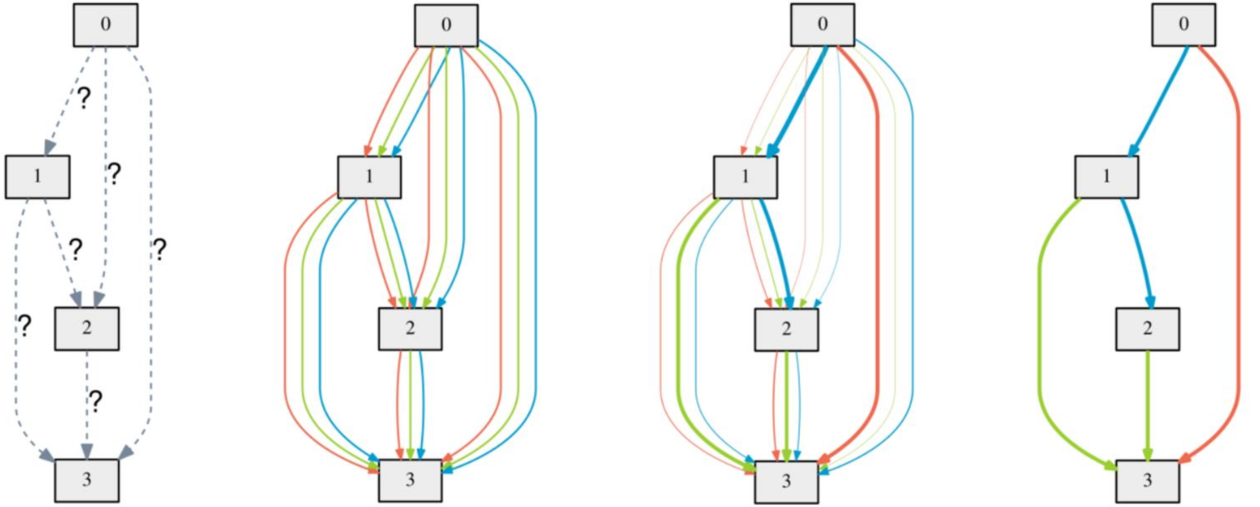
Learning cell = Learning edges

(which operations and which input nodes)



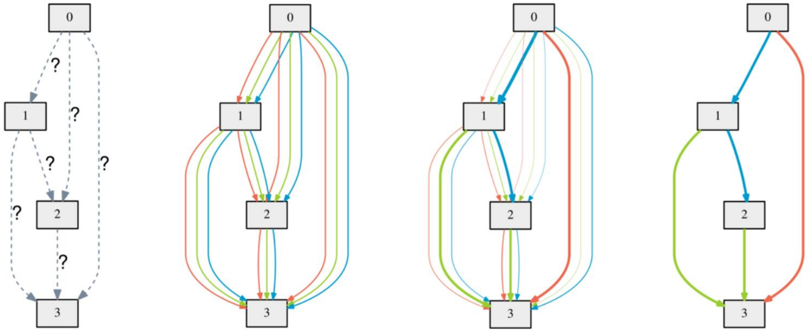
Search Space: Continuous Relaxation

Search Space: Continuous Relaxation



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

Search Space: Continuous Relaxation

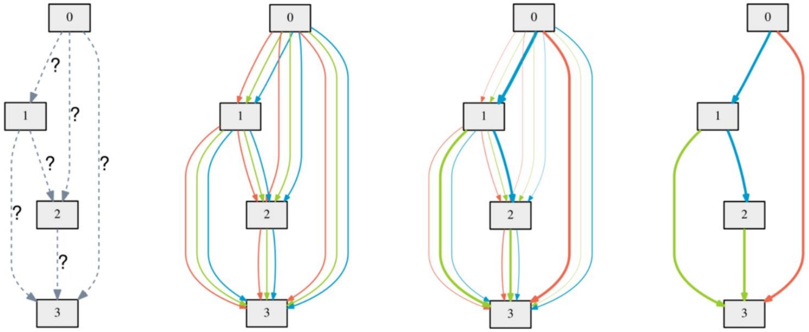


(a) (b) (c) (d)

Search Space: Continuous Relaxation

Mixture of operations through softmax:

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



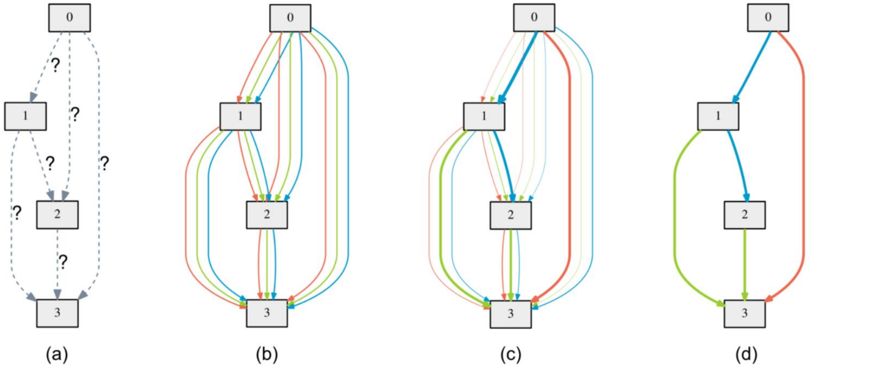
(a) (b) (c) (d)

Search Space: Continuous Relaxation

Mixture of operations through softmax:

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

O: set of candidate operations



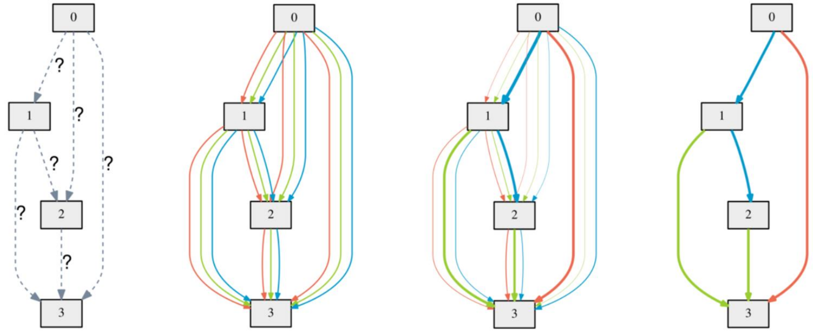
Search Space: Continuous Relaxation

Mixture of operations through softmax:

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

O: set of candidate operations

$o(x)$: function applied to latent representation x

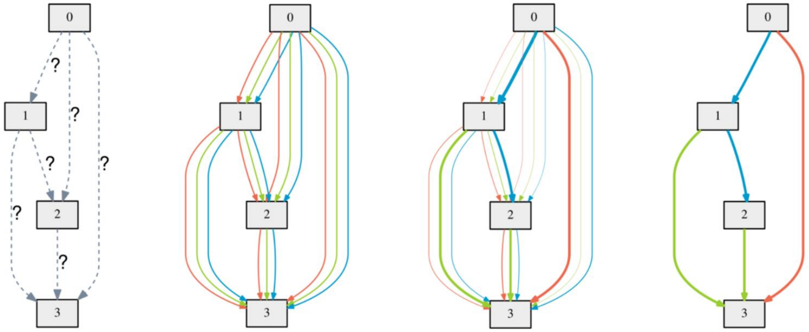


(a) (b) (c) (d)

Search Space: Continuous Relaxation

Mixture of operations through softmax:

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



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

O: set of candidate operations

$o(x)$: 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): {{3×3, 5×5} separable convolutions, {3×3, 5×5} dilated separable convolutions, 3×3 max pooling, 3×3 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): {{3×3, 5×5} separable convolutions, {3×3, 5×5} dilated separable convolutions, 3×3 max pooling, 3×3 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 α and weights w

Optimization

Goal: jointly learn architecture α and weights w

Bilevel Optimization:

Optimization

Goal: jointly learn architecture α and weights w

Bilevel Optimization:

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

Optimization

Goal: jointly learn architecture α 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 α 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. $w^*(\alpha) = \mathrm{argmin}_w \, \mathcal{L}_{train}(w, \alpha)$

Optimization

Goal: jointly learn architecture α 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. $w^*(\alpha) = \mathrm{argmin}_w \, \mathcal{L}_{train}(w, \alpha)$

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)

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

Optimization: Approximation

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

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

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:

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

Optimization: Approximation

Second order gradient leads to very large matrix vector multiplication:

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

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

Optimization: Approximation

Second order gradient leads to very large matrix vector multiplication:

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

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

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

where $w^{\pm} = w \pm \epsilon\ \nabla_{w'}\mathcal{L}_{val}(w',a)$

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 α 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 α.

Architecture Discretization

Architecture Discretization

Each node gets assigned the top-k strongest edges = largest α's

Architecture Discretization

Each node gets assigned the top-k strongest edges = largest α's

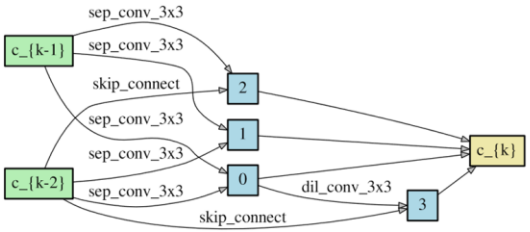
(k = 2, only nonzero operations)

Architecture Discretization

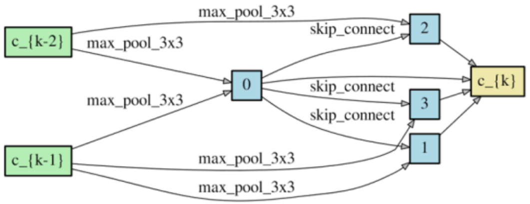
Each node gets assigned the top-k strongest edges = largest α's

(k = 2, only nonzero operations)

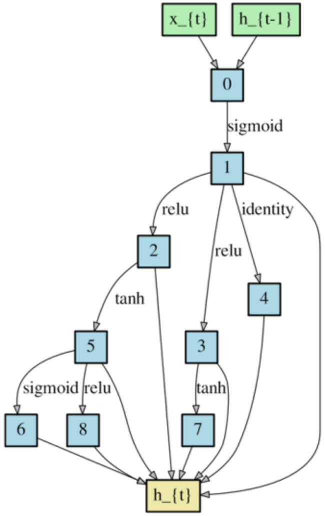
Resulting discrete cells:



Normal cell learned on CIFAR-10.



Reduction cell learned on CIFAR-10.



Recurrent cell learned on PTB.

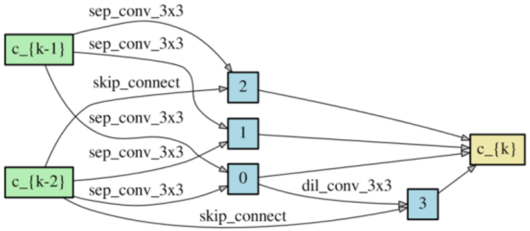
Architecture Discretization

Each node gets assigned the top-k strongest edges = largest α's

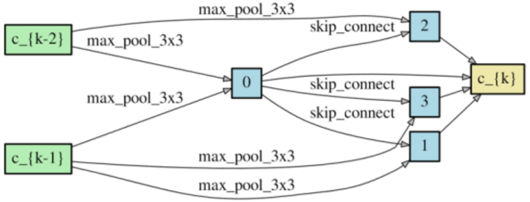
(k = 2, only nonzero operations)

Resulting discrete cells:

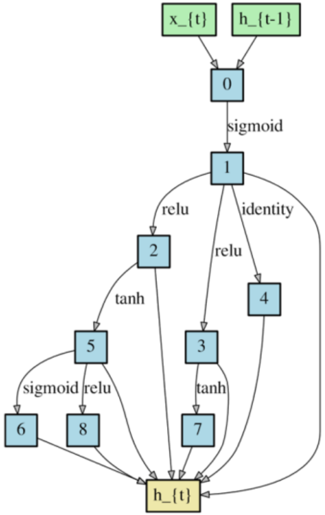
(get retrained, do not keep the w's)



Normal cell learned on CIFAR-10.



Reduction cell learned on CIFAR-10.



Recurrent cell learned on PTB.

Results

Results

- Random Search strong baseline
- Bilevel Optimization is essential

Results

- Random Search strong baseline
- Bilevel Optimization is essential

Architecture searched on CIFAR-10	CIFAR-10 Test Error (%)
Random Search	3.29 ± 0.15
DARTS (**Coordinate descent on all data**)	4.16 ± 0.16
DARTS (**Gradient descent on all data**)	3.56 ± 0.10
DARTS (**bilevel** optimization, **first order approximation**)	3.00 ± 0.14
DARTS (**bilevel** optimization, **second order approximation**)	2.76 ± 0.09

Results: CIFAR-10

Results: CIFAR-10

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

(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] → multi-objective optimization: maximize accuracy and minimize FLOPS

Conclusion - further work in NAS

RL and evolution direction:

MnasNet [5] → multi-objective optimization: maximize accuracy and minimize FLOPS
Was used for *EfficientNet* [6]

Questions?

References

[1] DARTS: Differentiable Architecture Search. Liu et al. (2018).

[2] Neural Architecture Search with Reinforcement Learning. Zoph et al. (2016).

[3] Learning Transferable Architectures for Scalable Image Recognition. Zoph et al. (2017).

[4] Regularized Evolution for Image Classifier Architecture Search. Real et al. (2018).

[5] MnasNet: Platform-Aware Neural Architecture Search for Mobile. Tan et al. (2019).

[6] EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Tan, Le (2019).

[7] Progressive Differentiable Architecture Search: Bridging the Depth Gap between Search and Evaluation. Chen et al. (2019).

[8] Fair DARTS: Eliminating Unfair Advantages in Differentiable Architecture Search. Chu et al. (2020).

[9] DARTS+: Improved Differentiable Architecture Search with Early Stopping. Liang et al. (2019).

[10] sharpDARTS: Faster and More Accurate Differentiable Architecture Search. Hundt et al. (2019).

[11] PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. Xu et al. (2020).

[12] Are Labels Necessary for Neural Architecture Search?. Liu et al. (2020).

[13] ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. Cai et al. (2019).