Neural Architecture Search

Neural Networks Design And Application

A framework for NAS

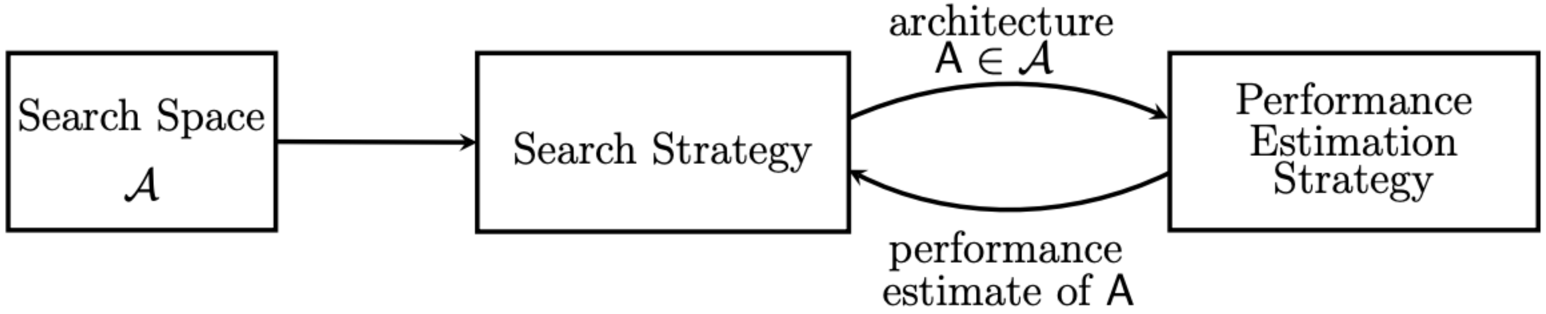
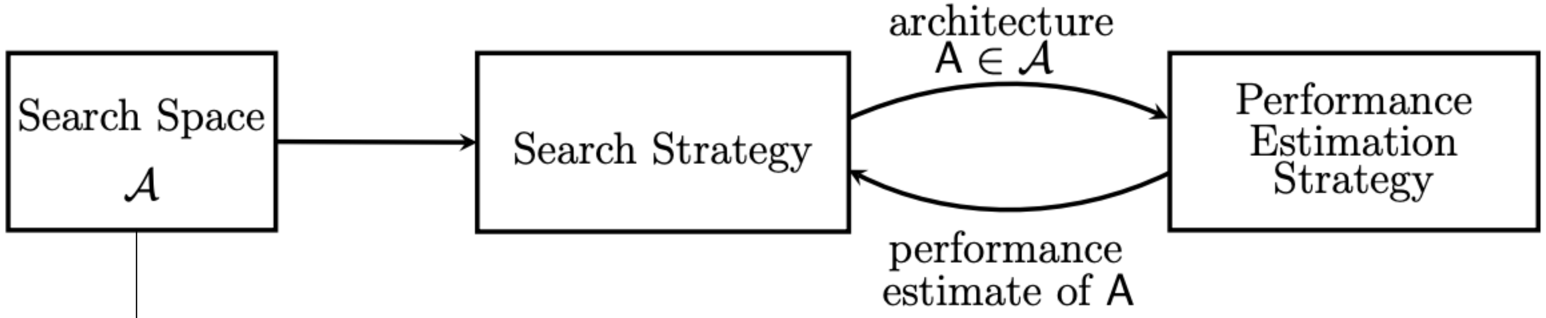
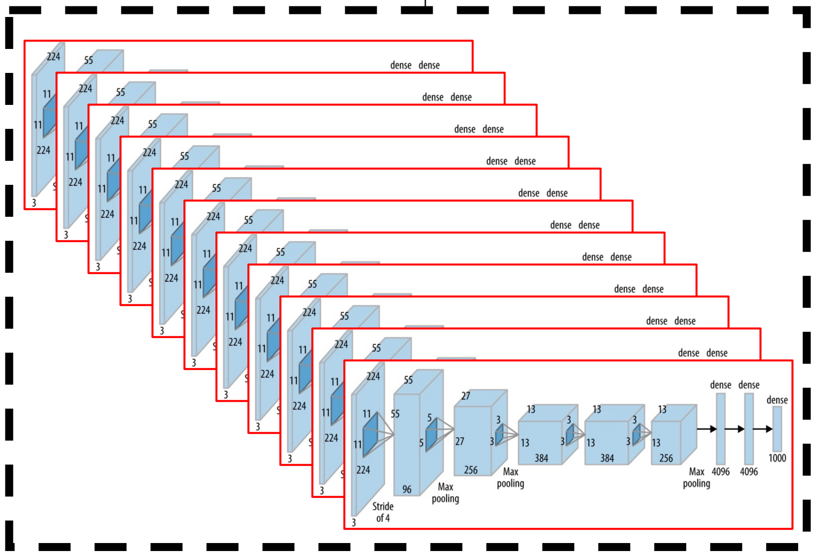


Image credit:
Elsken, Thomas, Jan Hendrik Metzen, and Frank Hutter. "Neural architecture search: A survey." *J. Mach. Learn. Res.* 20, no. 55 (2019): 1-21.
https://arxiv.org/pdf/1808.05377.pdf

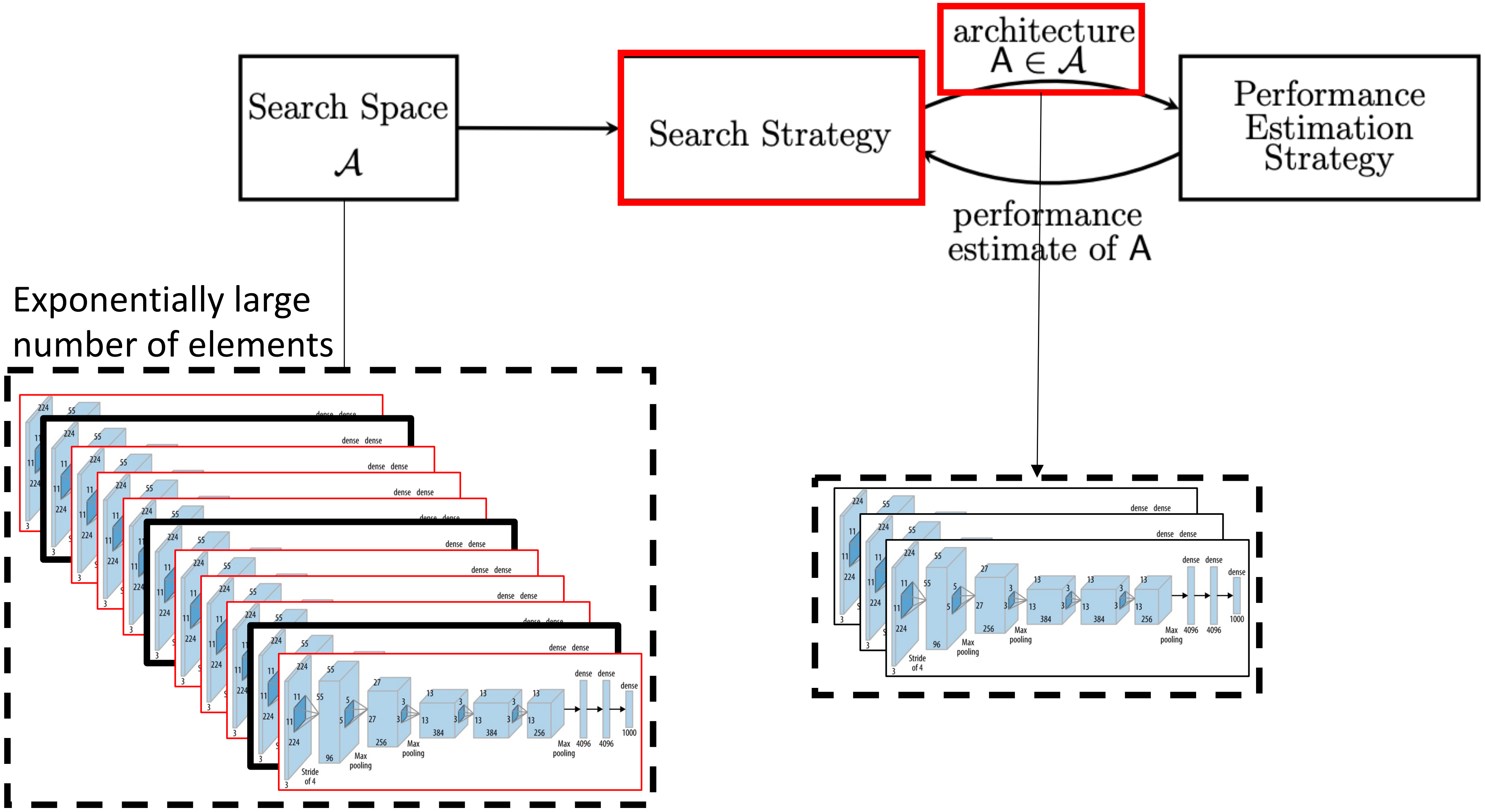
A framework for NAS



Exponentially large
number of elements



A framework for NAS

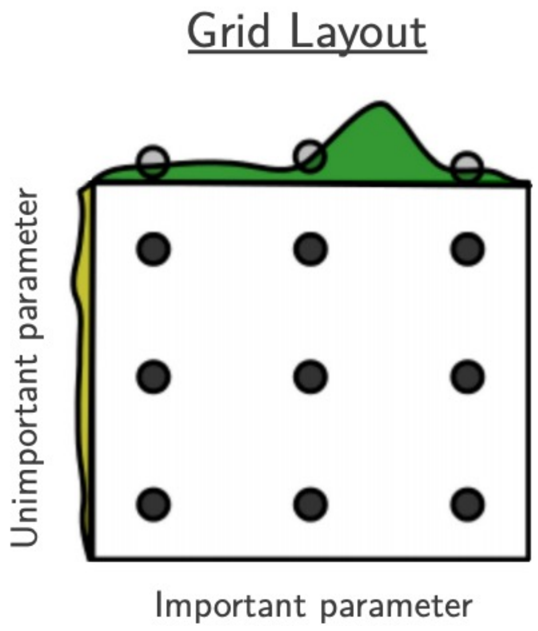


Random search

- Learning rate: {0.0001, 0.001, 0.01, 0.1, 1, 10}
- Weight decay (L2 regularization): {0.0001, 0.001, 0.01, 0.1, 1, 10}

Random search

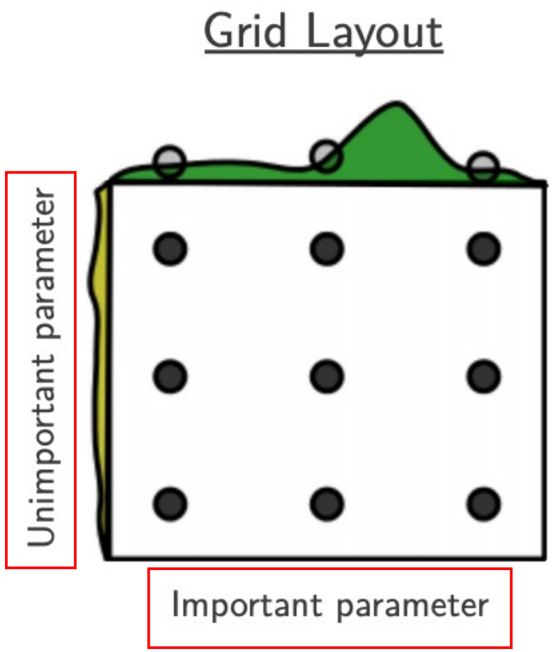
- Learning rate: {0.0001, 0.001, 0.01, 0.1, 1, 10}
- Weight decay (L2 regularization): {0.0001, 0.001, 0.01, 0.1, 1, 10}



https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Random search

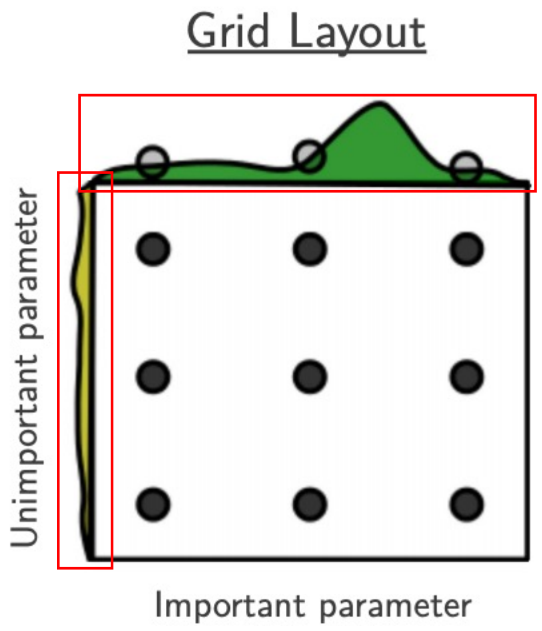
- Learning rate: {0.0001, 0.001, 0.01, 0.1, 1, 10}
- Weight decay (L2 regularization): {0.0001, 0.001, 0.01, 0.1, 1, 10}



https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Random search

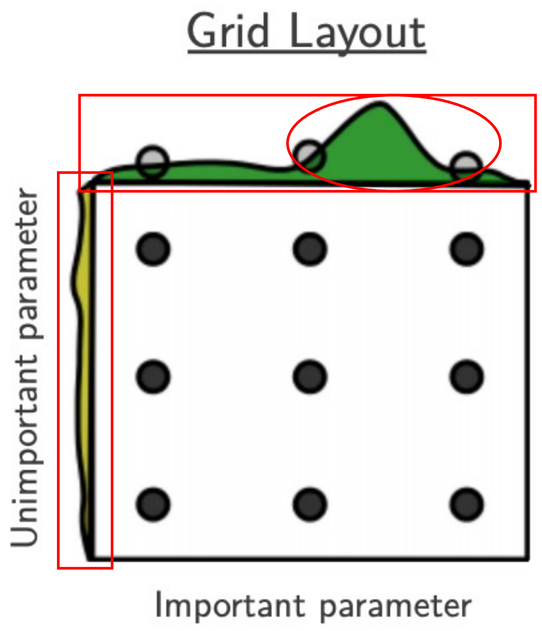
- Learning rate: {0.0001, 0.001, 0.01, 0.1, 1, 10}
- Weight decay (L2 regularization): {0.0001, 0.001, 0.01, 0.1, 1, 10}



https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Random search

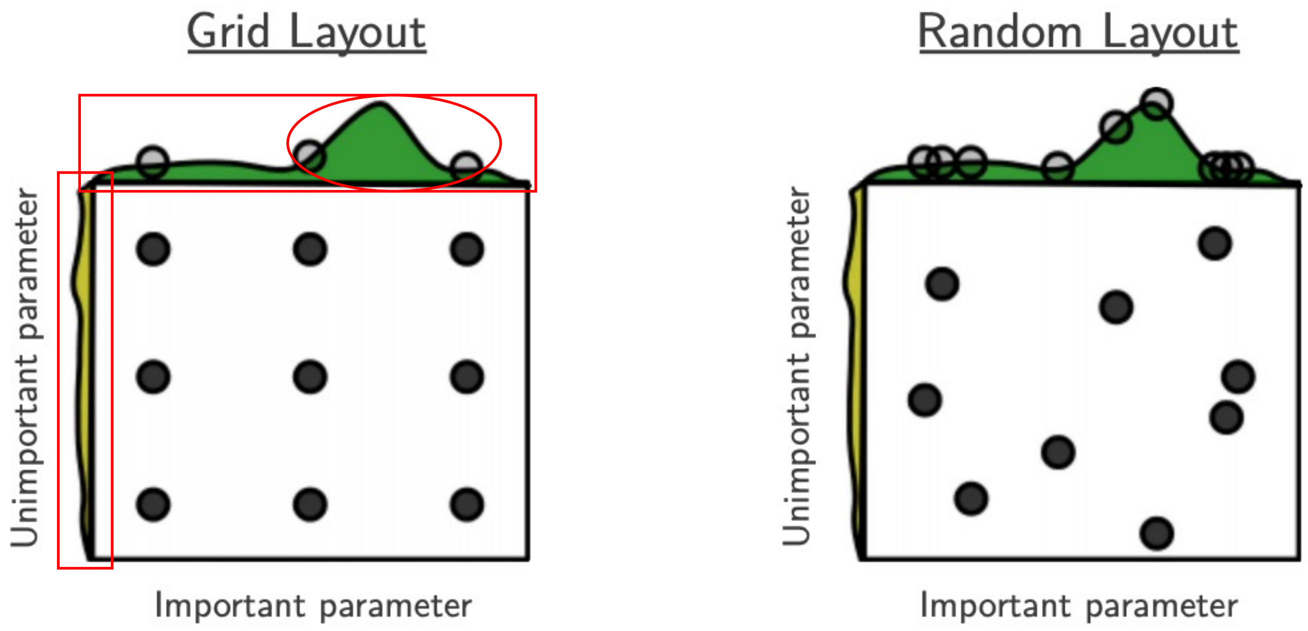
- Learning rate: {0.0001, 0.001, 0.01, 0.1, 1, 10}
- Weight decay (L2 regularization): {0.0001, 0.001, 0.01, 0.1, 1, 10}



https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Random search

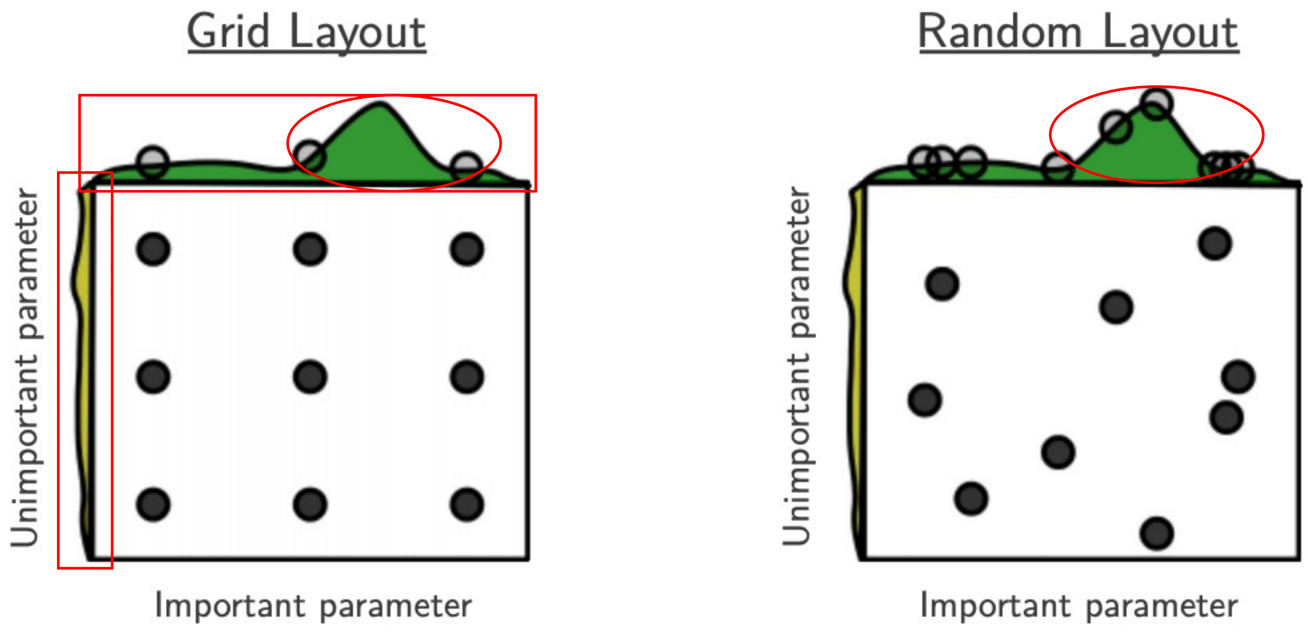
- Learning rate: {0.0001, 0.001, 0.01, 0.1, 1, 10}
- Weight decay (L2 regularization): {0.0001, 0.001, 0.01, 0.1, 1, 10}



https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Random search

- Learning rate: {0.0001, 0.001, 0.01, 0.1, 1, 10}
- Weight decay (L2 regularization): {0.0001, 0.001, 0.01, 0.1, 1, 10}



https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Reinforcement learning

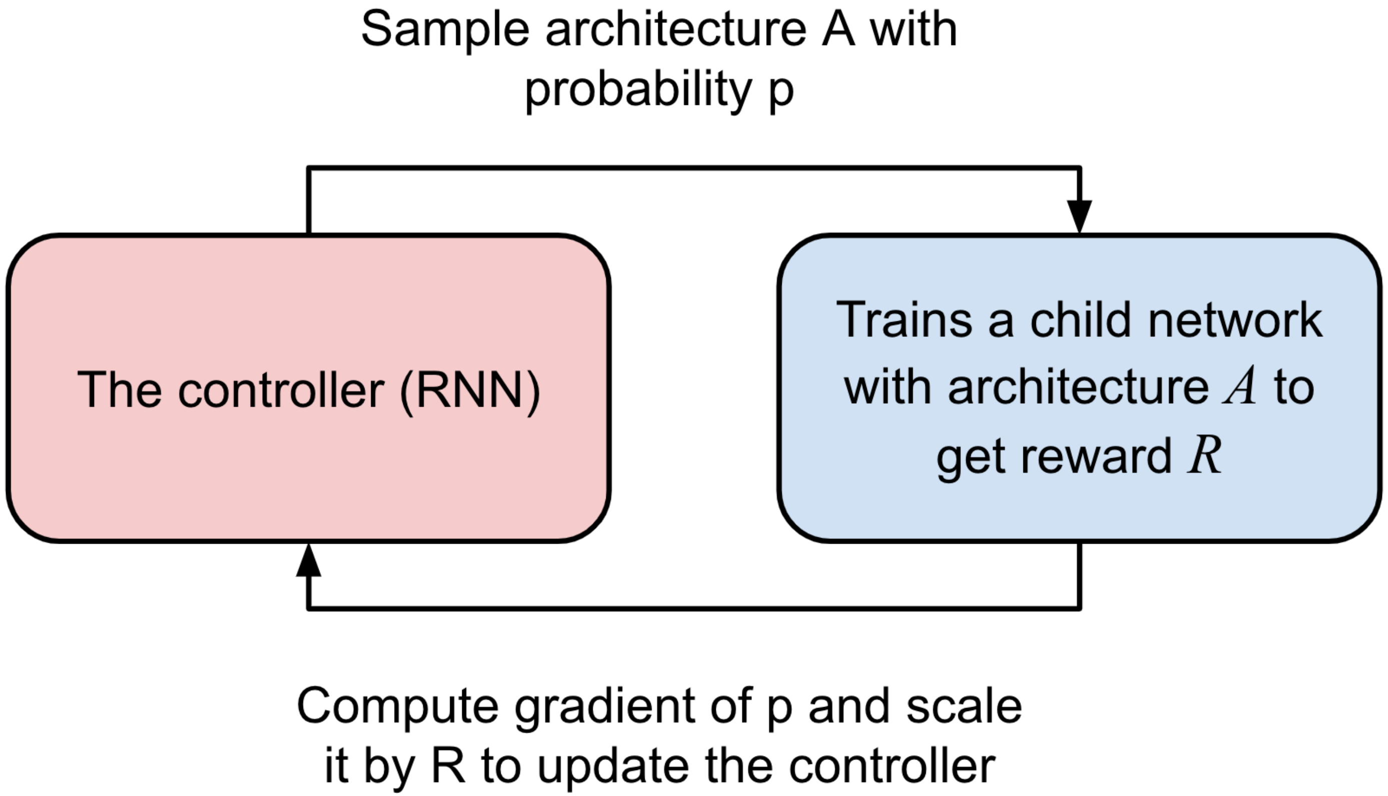


Image credit: https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

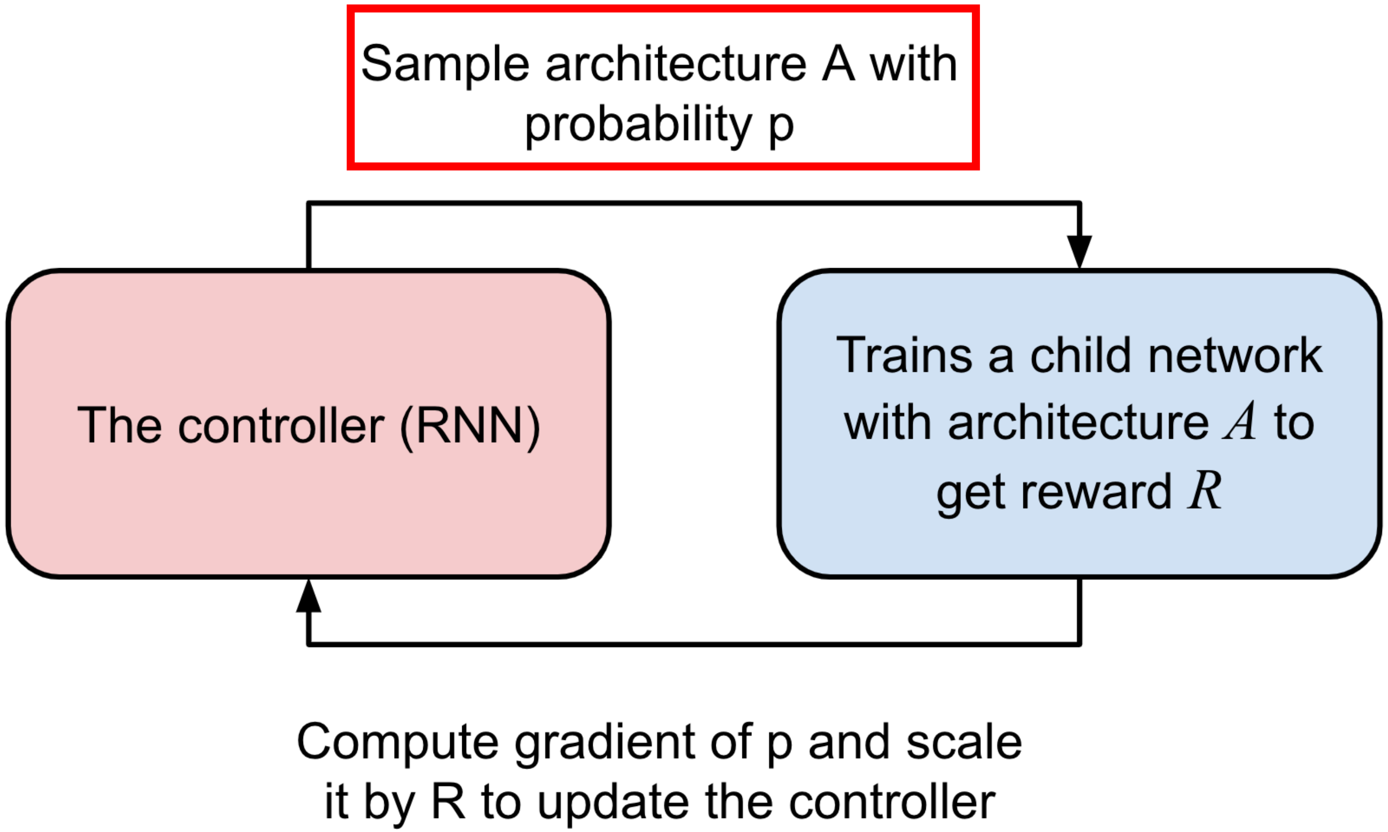
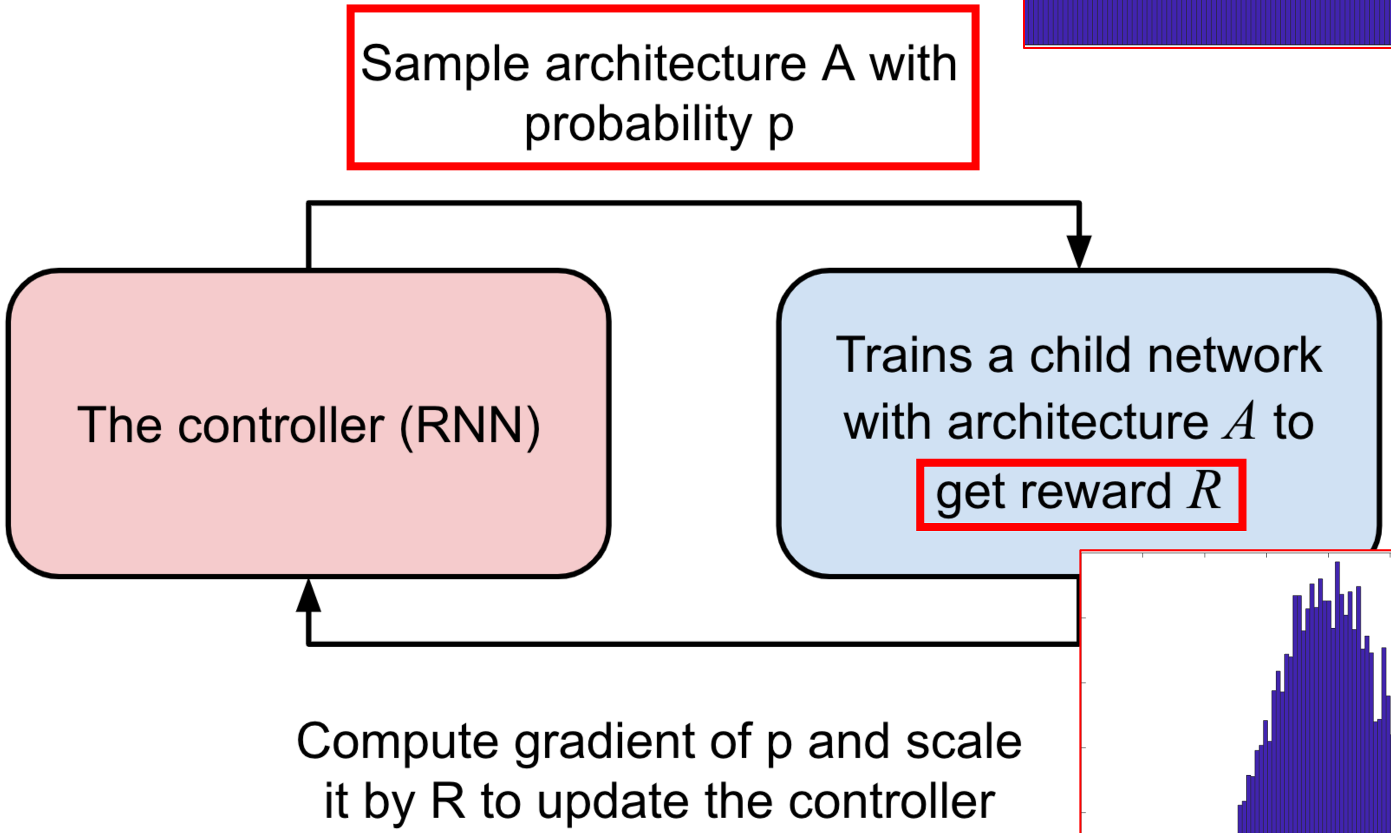
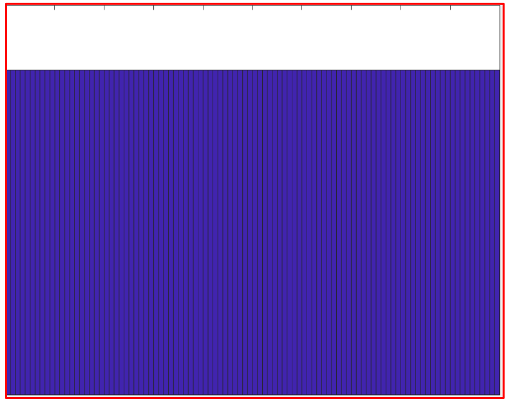
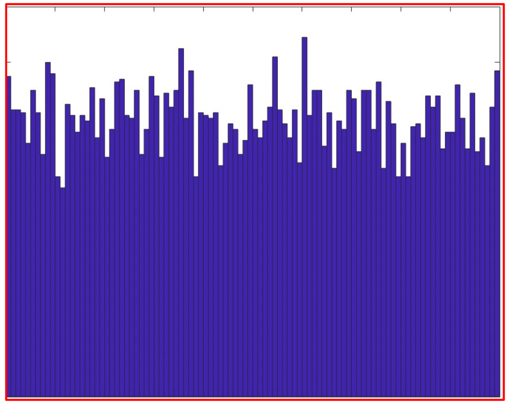


Image credit: https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning







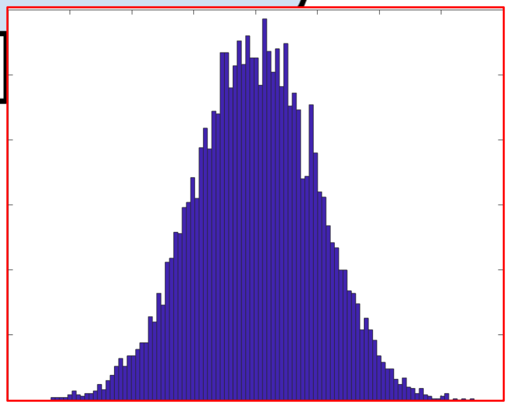


Image credit: https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

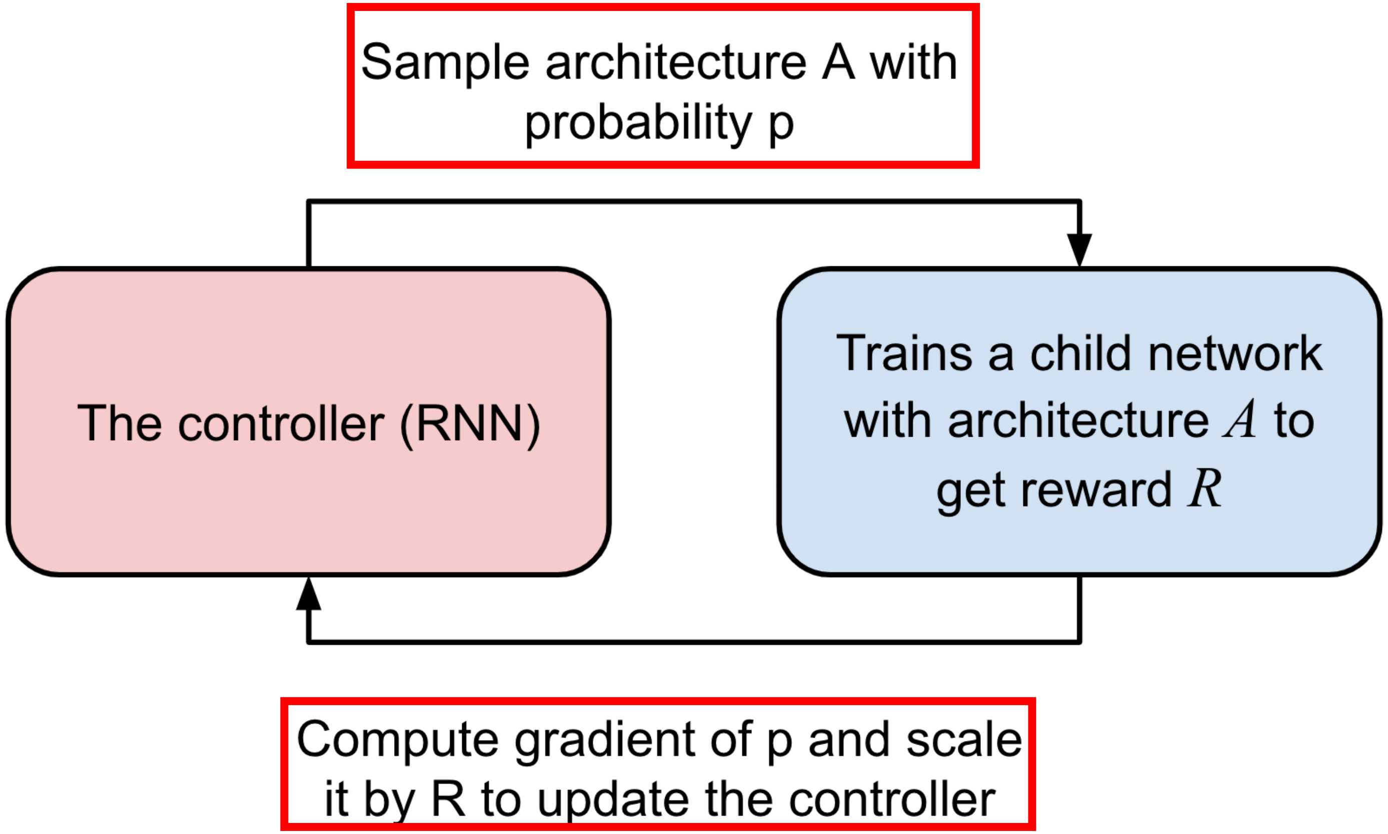


Image credit: https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

- **Action space**: The action space is a list of tokens for defining a child network predicted by the controller (See more in the above section). The controller outputs *action*, $a_{1:T}$, where T is the total number of tokens.
- **Reward**: The accuracy of a child network that can be achieved at convergence is the reward for training the controller, R.
- **Loss**: NAS optimizes the controller parameters θ with a REINFORCE loss. We want to maximize the expected reward (high accuracy) with the gradient as follows. The nice thing here with policy gradient is that it works even when the reward is non-differentiable.

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log P(a_t | a_{1:(t-1)}; \theta) R]$$

https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

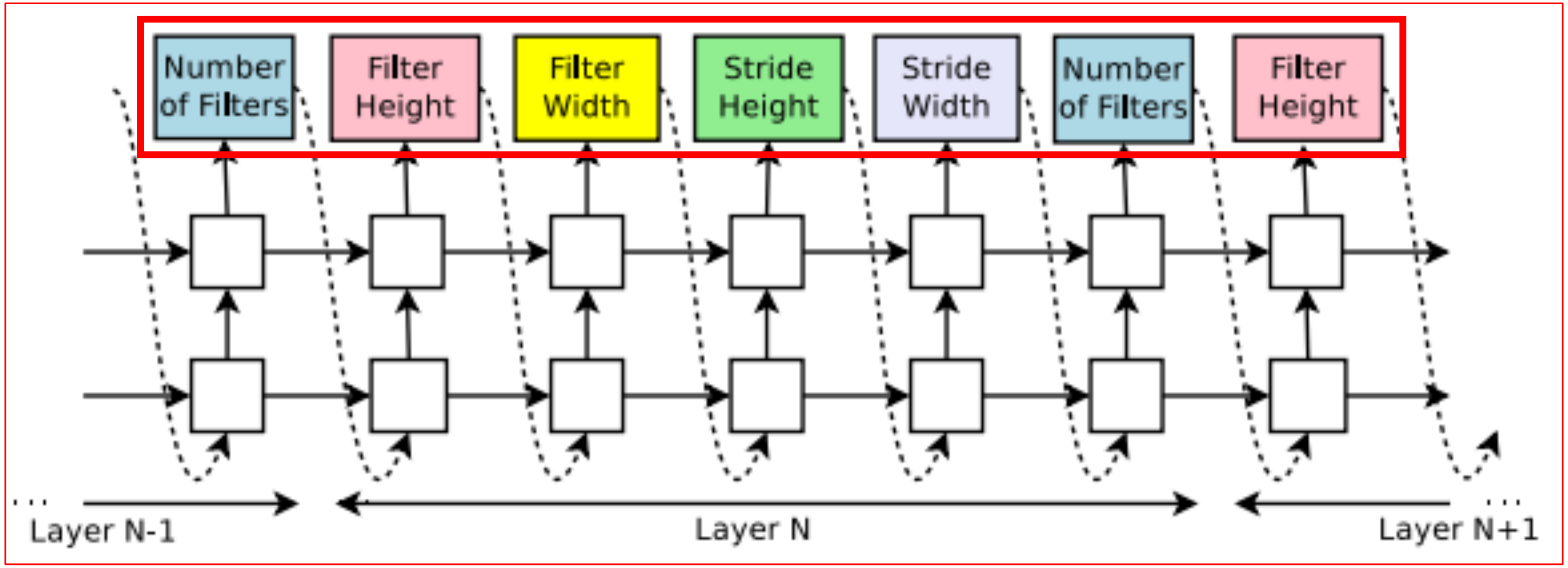
Reinforcement learning

- **Action space** The action space is a list of tokens for defining a child network predicted by the controller (See more in the above section). The controller outputs *action*, $a_{1:T}$, where T is the total number of tokens.
- **Reward**: The accuracy of a child network that can be achieved at convergence is the reward for training the controller, R.
- **Loss**: NAS optimizes the controller parameters θ with a REINFORCE loss. We want to maximize the expected reward (high accuracy) with the gradient as follows. The nice thing here with policy gradient is that it works even when the reward is non-differentiable.

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log P(a_t | a_{1:(t-1)}; \theta) R]$$

https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Defining search space



A RNN network

Zoph, Barret, and Quoc V. Le. "Neural architecture search with reinforcement learning." *arXiv preprint arXiv:1611.01578* (2016). https://arxiv.org/pdf/1611.01578.pdf

Reinforcement learning

- **Action space** The action space is a list of tokens for defining a child network predicted by the controller (See more in the above section). The controller outputs $\boxed{action,\ a_{1:T}}$ where $\boxed{T}$ is the total number of tokens.
- **Reward**: The accuracy of a child network that can be achieved at convergence is the reward for training the controller, R.
- **Loss**: NAS optimizes the controller parameters θ with a REINFORCE loss. We want to maximize the expected reward (high accuracy) with the gradient as follows. The nice thing here with policy gradient is that it works even when the reward is non-differentiable.

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log P(a_t | a_{1:(t-1)}; \theta) R]$$

https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

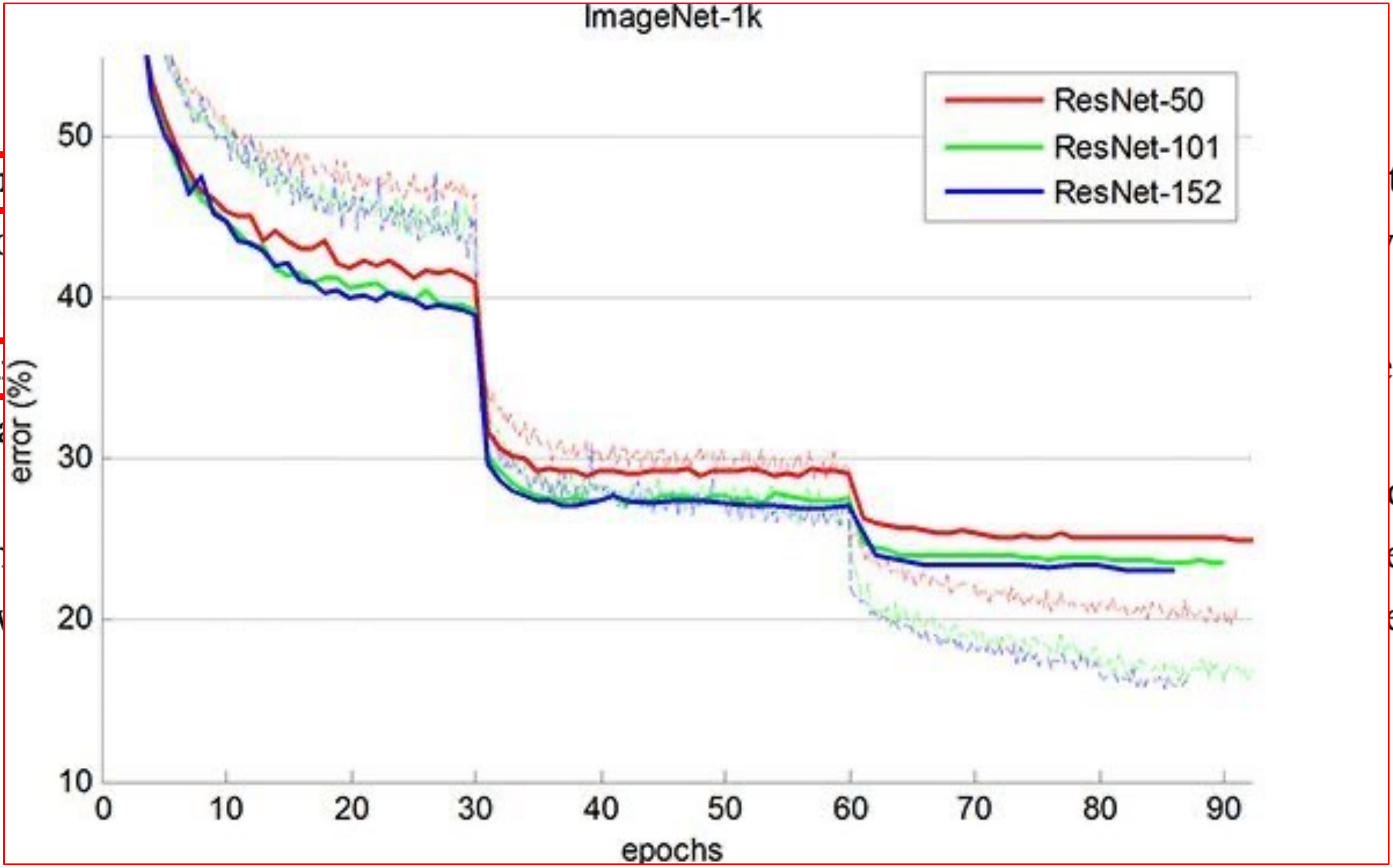
- **Action space** The action space is a list of tokens for defining a child network predicted by the controller (See more in the above section). The controller outputs *action*, $a_{1:T}$ where T is the total number of tokens.
- **Reward**: The accuracy of a child network that can be achieved at convergence is the reward for training the controller, R.
- **Loss**: NAS optimizes the controller parameters θ with a REINFORCE loss. We want to maximize the expected reward (high accuracy) with the gradient as follows. The nice thing here with policy gradient is that it works even when the reward is non-differentiable.

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log P(a_t | a_{1:(t-1)}; \theta) R]$$

https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

- **Actio** ... ted by
  the co ... here T
  is the
- **Rewa** ... reward
  for tra
- **Loss:** ... o
  maxin ... e thing
  here v ... e.



https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

- **Action space**: The action space is a list of tokens for defining a child network predicted by the controller (See more in the above section). The controller outputs $action,\ a_{1:T}$ where T is the total number of tokens.
- **Reward**: The accuracy of a child network that can be achieved at convergence is the reward for training the controller, R.
- **Loss**: NAS optimizes the controller parameters θ with a REINFORCE loss. We want to maximize the expected reward (high accuracy) with the gradient as follows. The nice thing here with policy gradient is that it works even when the reward is non-differentiable.

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log P(a_t|a_{1:(t-1)};\theta)R]$$

https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

- **Action space** The action space is a list of tokens for defining a child network predicted by the controller (See more in the above section). The controller outputs $action, a_{1:T}$ where T is the total number of tokens.
- **Reward**: The accuracy of a child network that can be achieved at convergence is the reward for training the controller, R.
- **Loss**: NAS optimizes the controller parameters θ with a REINFORCE loss. We want to maximize the expected reward (high accuracy) with the gradient as follows. The nice thing here with policy gradient is that it works even when the reward is non-differentiable.

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log P(a_t | a_{1:(t-1)}; \theta) R]$$

https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

- **Action space**: The action space is a list of tokens for defining a child network predicted by the controller (See more in the above section). The controller outputs $\boxed{action,\ a_{1:T}}$ where $\boxed{T}$ is the total number of tokens.
- **Reward**: The accuracy of a child network that can be achieved at convergence is the reward for training the controller, $\boxed{R}$.
- **Loss**: NAS optimizes the $\boxed{\text{controller parameters } \theta}$ with a REINFORCE loss. We want to maximize the expected reward (high accuracy) with the gradient as follows. The nice thing here with policy gradient is that it works even when the reward is non-differentiable.

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log P(a_t | a_{1:(t-1)}; \theta) R]$$

https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

- **Action space**: The action space is a list of tokens for defining a child network predicted by the controller (See more in the above section). The controller outputs *action*, $a_{1:T}$ where T is the total number of tokens.
- **Reward**: The accuracy of a child network that can be achieved at convergence is the reward for training the controller, R.
- **Loss**: NAS optimizes the controller parameters θ with a REINFORCE loss. We want to maximize the expected reward (high accuracy) with the gradient as follows. The nice thing here with policy gradient is that it works even when the reward is non-differentiable.

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log P(a_t | a_{1:(t-1)}; \theta) R]$$

https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

Reinforcement learning

- **Action space**: The action space is a list of tokens for defining a child network predicted by the controller (See more in the above section). The controller outputs *action*, $a_{1:T}$ where T is the total number of tokens.
- **Reward**: The accuracy of a child network that can be achieved at convergence is the reward for training the controller, R.
- **Loss**: NAS optimizes the controller parameters θ with a REINFORCE loss. We want to maximize the expected reward (high accuracy) with the gradient as follows. The nice thing here with policy gradient is that it works even when the reward is non-differentiable.

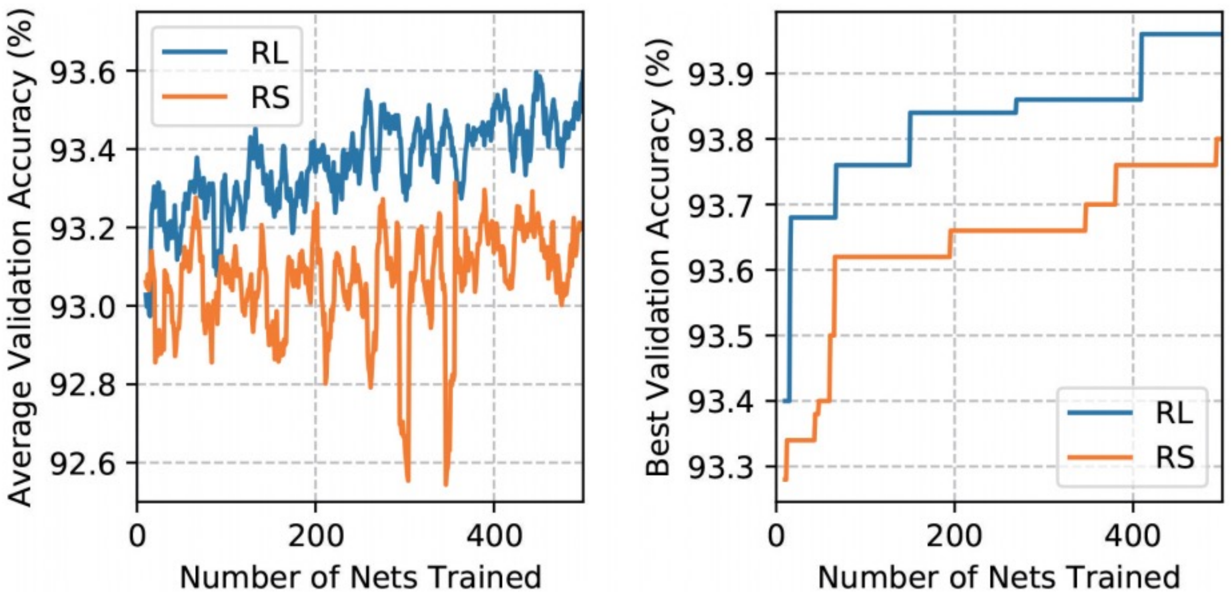
$$\nabla_\theta J(\theta) = \boxed{\sum_{t=1}^{T} \mathbb{E}[\nabla_\theta \log P(a_t|a_{1:(t-1)};\theta)R]}$$

$$\boxed{\mathrm{E}[X] = \sum_{i=1}^{k} x_i\, p_i = x_1p_1 + x_2p_2 + \cdots + x_kp_k.}$$

$$\boxed{L(\theta \mid x) = \prod_{i=1}^{n} p_X(x_i \mid \theta)}$$

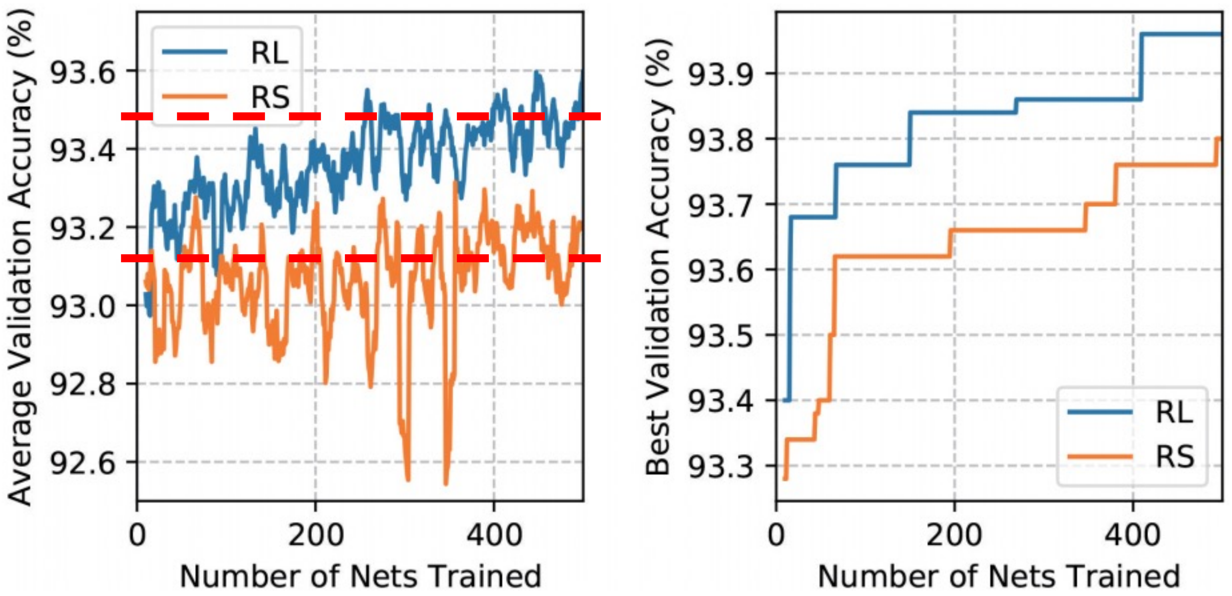
https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html

RS vs RL



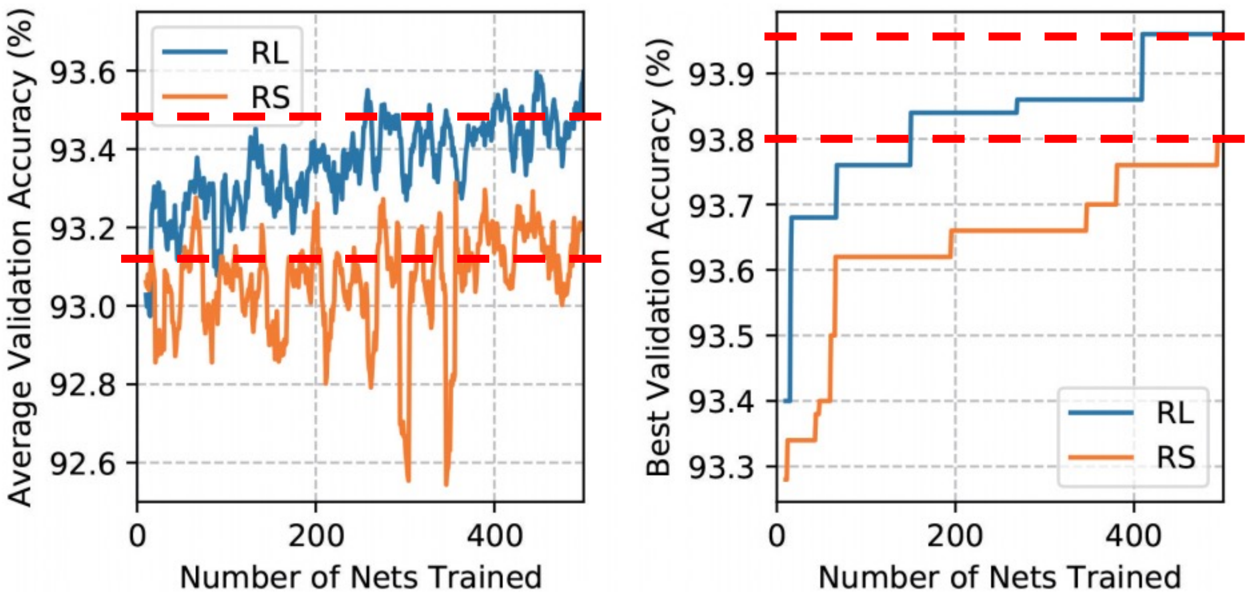
Cai, Han, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. "Path-level network transformation for efficient architecture search." In *International Conference on Machine Learning*, pp. 678-687. PMLR, 2018. https://arxiv.org/pdf/1806.02639.pdf

RS vs RL



Cai, Han, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. "Path-level network transformation for efficient architecture search." In *International Conference on Machine Learning*, pp. 678-687. PMLR, 2018. https://arxiv.org/pdf/1806.02639.pdf

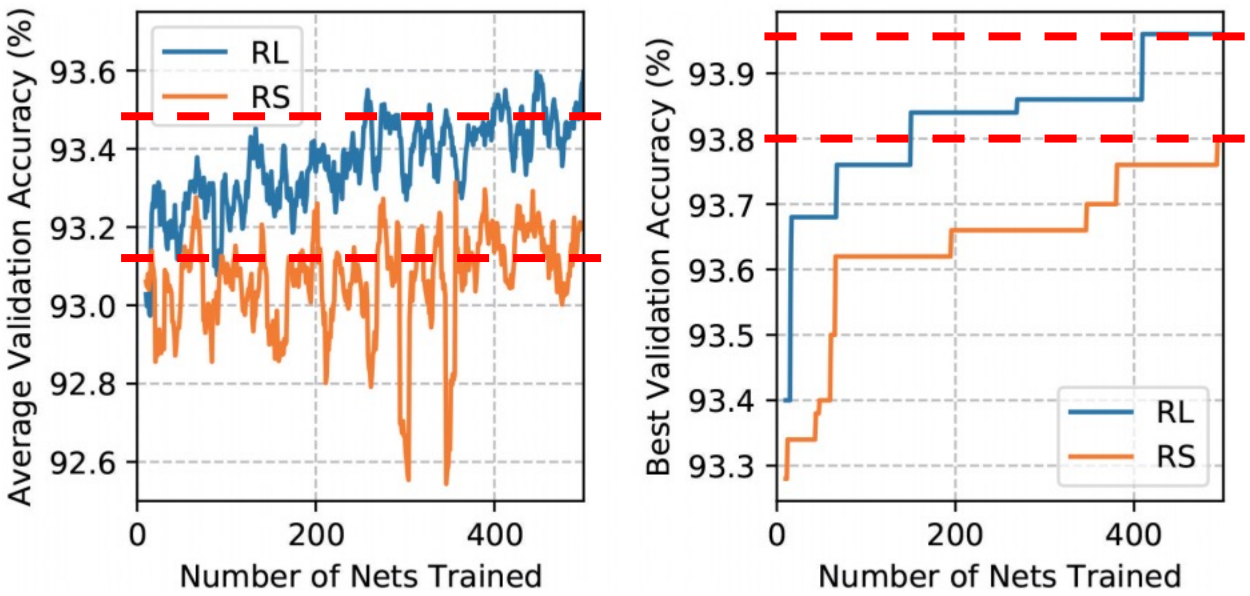
RS vs RL



Cai, Han, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. "Path-level network transformation for efficient architecture search." In *International Conference on Machine Learning*, pp. 678-687. PMLR, 2018.
https://arxiv.org/pdf/1806.02639.pdf

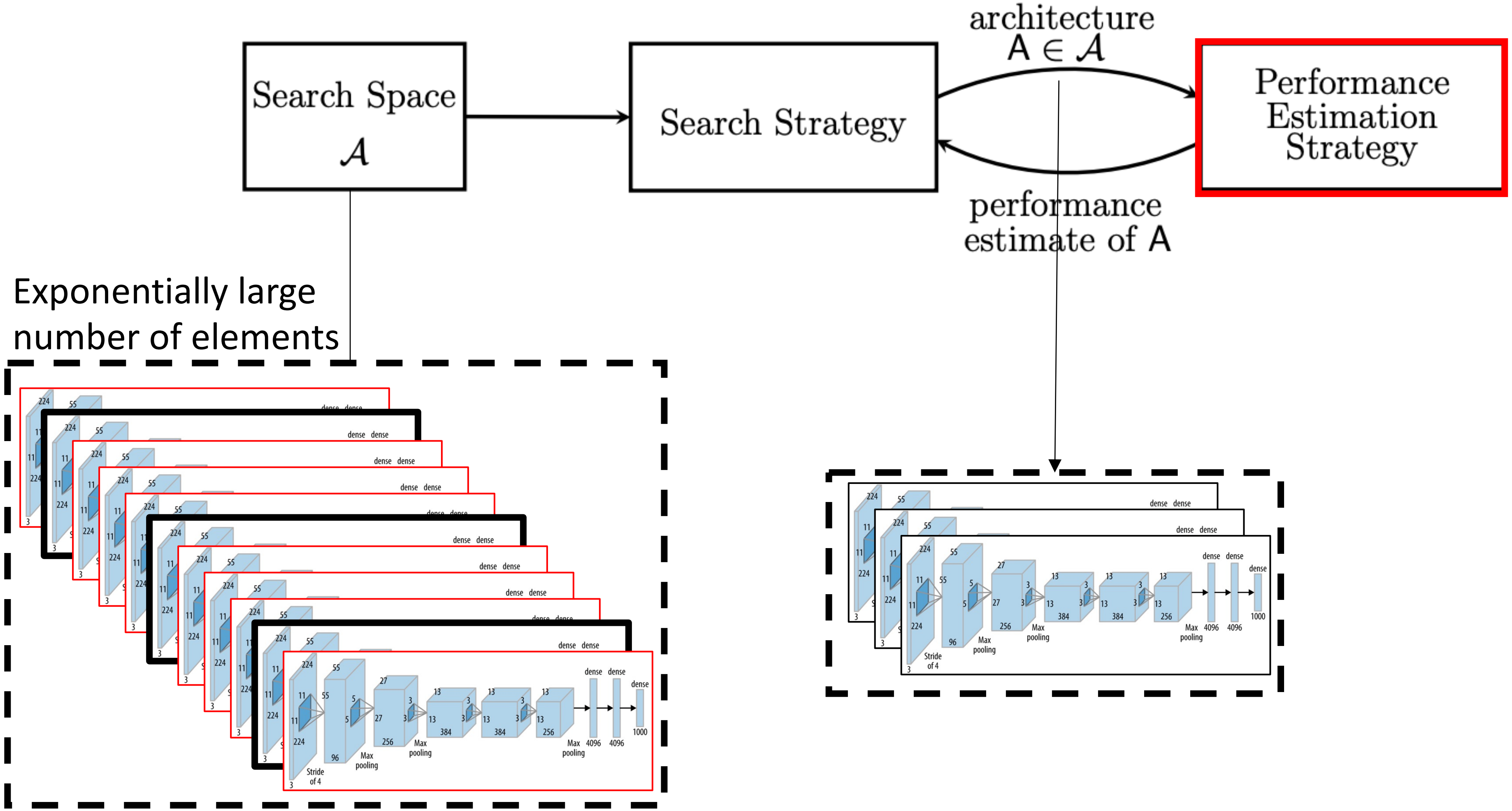
RS vs RL

Q: what conclusion can we make?



Cai, Han, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. "Path-level network transformation for efficient architecture search." In *International Conference on Machine Learning*, pp. 678-687. PMLR, 2018.
https://arxiv.org/pdf/1806.02639.pdf

A framework for NAS



Evaluation Strategy

- Training from scratch (until convergence)

Evaluation Strategy

- Training from scratch (until convergence)
- Proxy Task Performance

Evaluation Strategy

- Training from scratch (until convergence)
- Proxy Task Performance
  - Train on a smaller dataset.

Evaluation Strategy

- Training from scratch (until convergence)
- Proxy Task Performance
  - Train on a smaller dataset.
  - Train for fewer epochs.

Evaluation Strategy

- Training from scratch (until convergence)
- Proxy Task Performance
  - Train on a smaller dataset.
  - Train for fewer epochs.
  - Train and evaluate a down-scaled model in the search stage

Evaluation Strategy

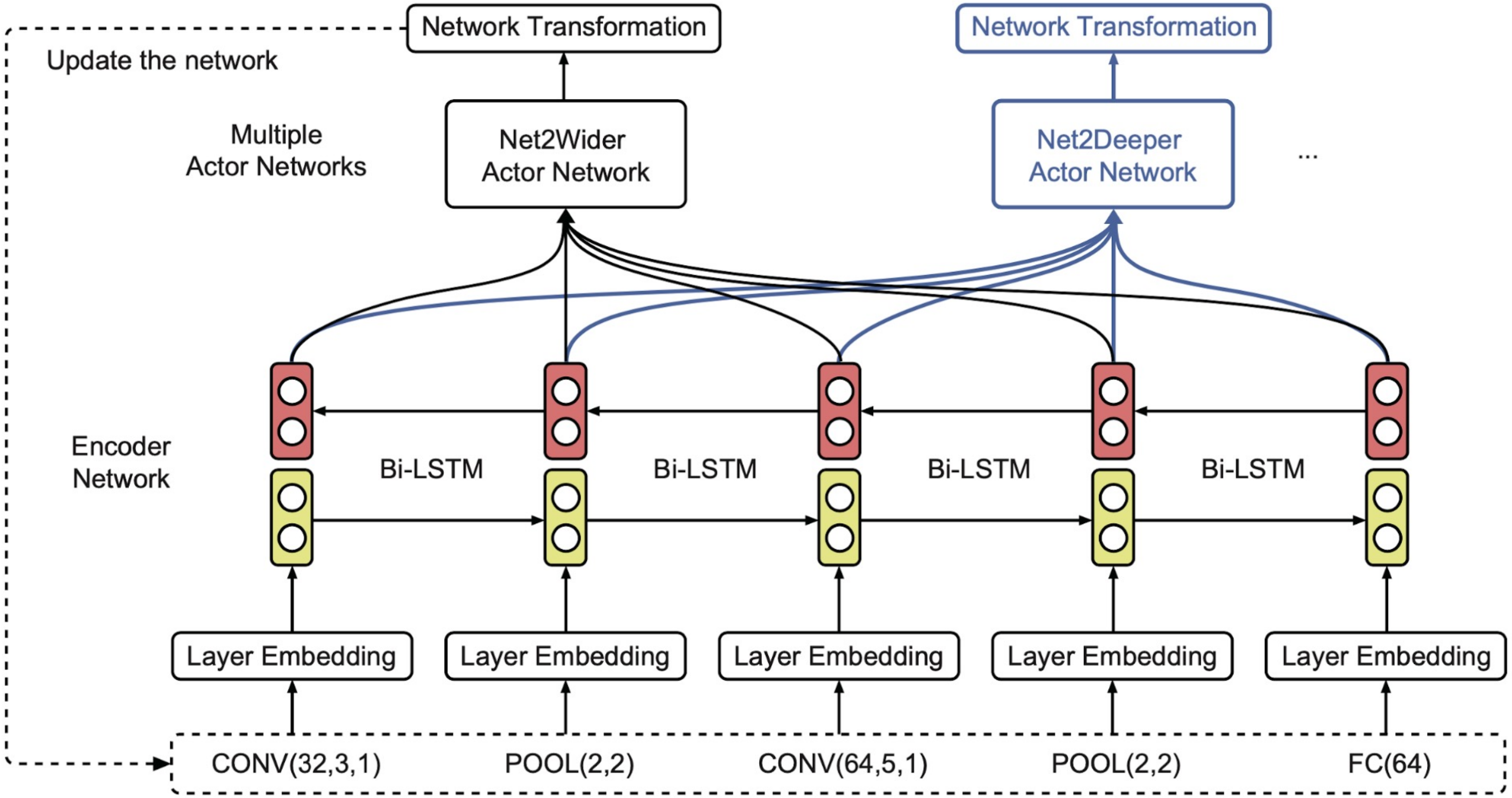
- Training from scratch (until convergence)
- Proxy Task Performance
  - Train on a smaller dataset.
  - Train for fewer epochs.
  - Train and evaluate a down-scaled model in the search stage
    - E.g., once a cell structure is learned, the final architecture is determined by selecting a number of cell repeats https://arxiv.org/pdf/1707.07012.pdf

putational budget. After having learned the convolutional cells, several hyper-parameters may be explored to build a final network for a given task: (1) the number of cell repeats N and (2) the number of filters in the initial convolutional cell. After selecting the number of initial filters, we use a

Evaluation Strategy

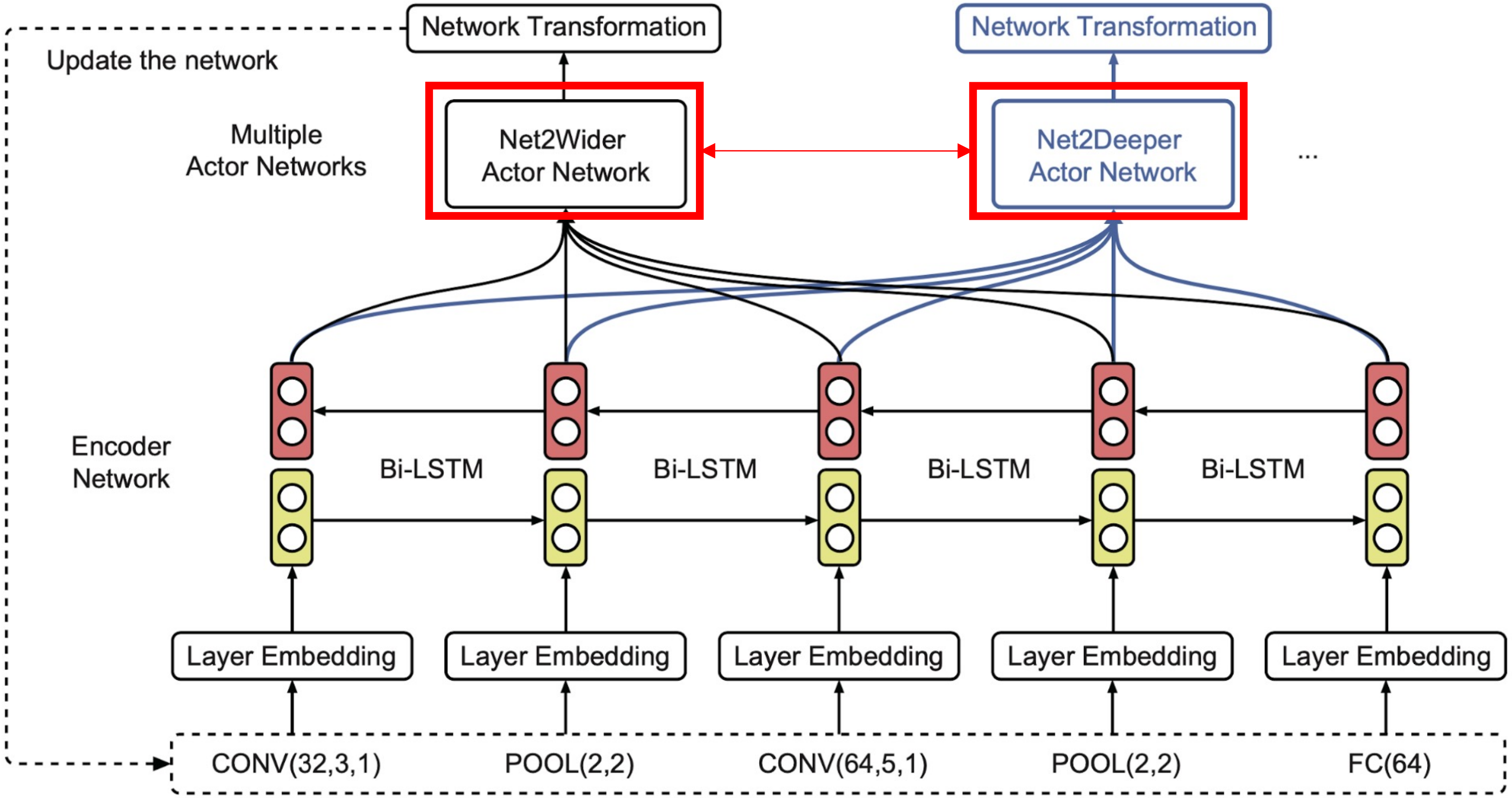
- Training from scratch (until convergence)
- Proxy Task Performance
  - Train on a smaller dataset.
  - Train for fewer epochs.
  - Train and evaluate a down-scaled model in the search stage
    - E.g., once a cell structure is learned, the final architecture is determined by selecting a number of cell repeats https://arxiv.org/pdf/1707.07012.pdf
- Parameter Sharing

Parameter Sharing



Cai, Han, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. "Efficient architecture search by network transformation." In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1. 2018.
https://arxiv.org/pdf/1707.04873.pdf

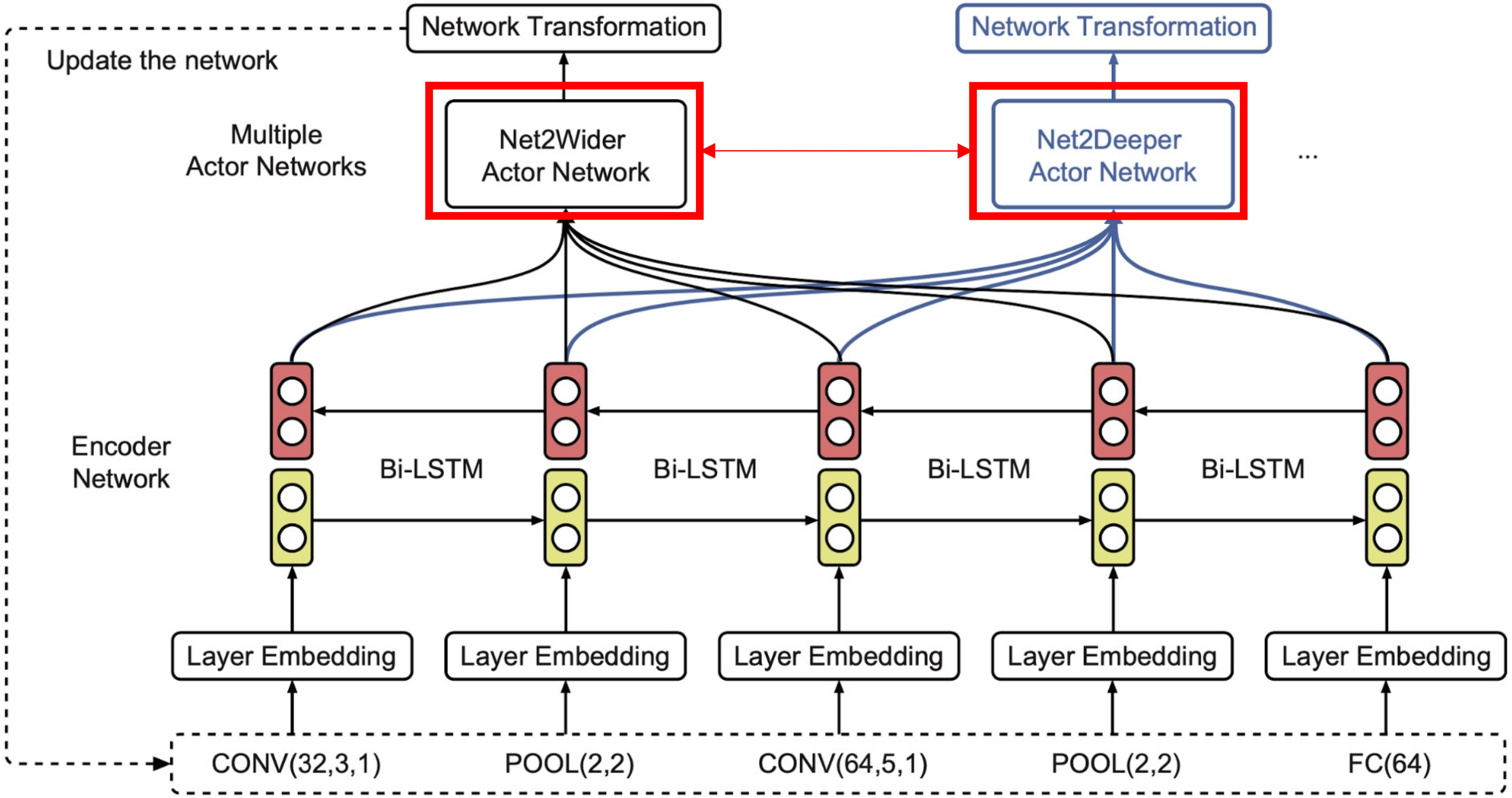
Parameter Sharing



Cai, Han, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. "Efficient architecture search by network transformation." In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1. 2018.
https://arxiv.org/pdf/1707.04873.pdf

Parameter Sharing

more units for fully-connected layers
more filters for convolutional layers

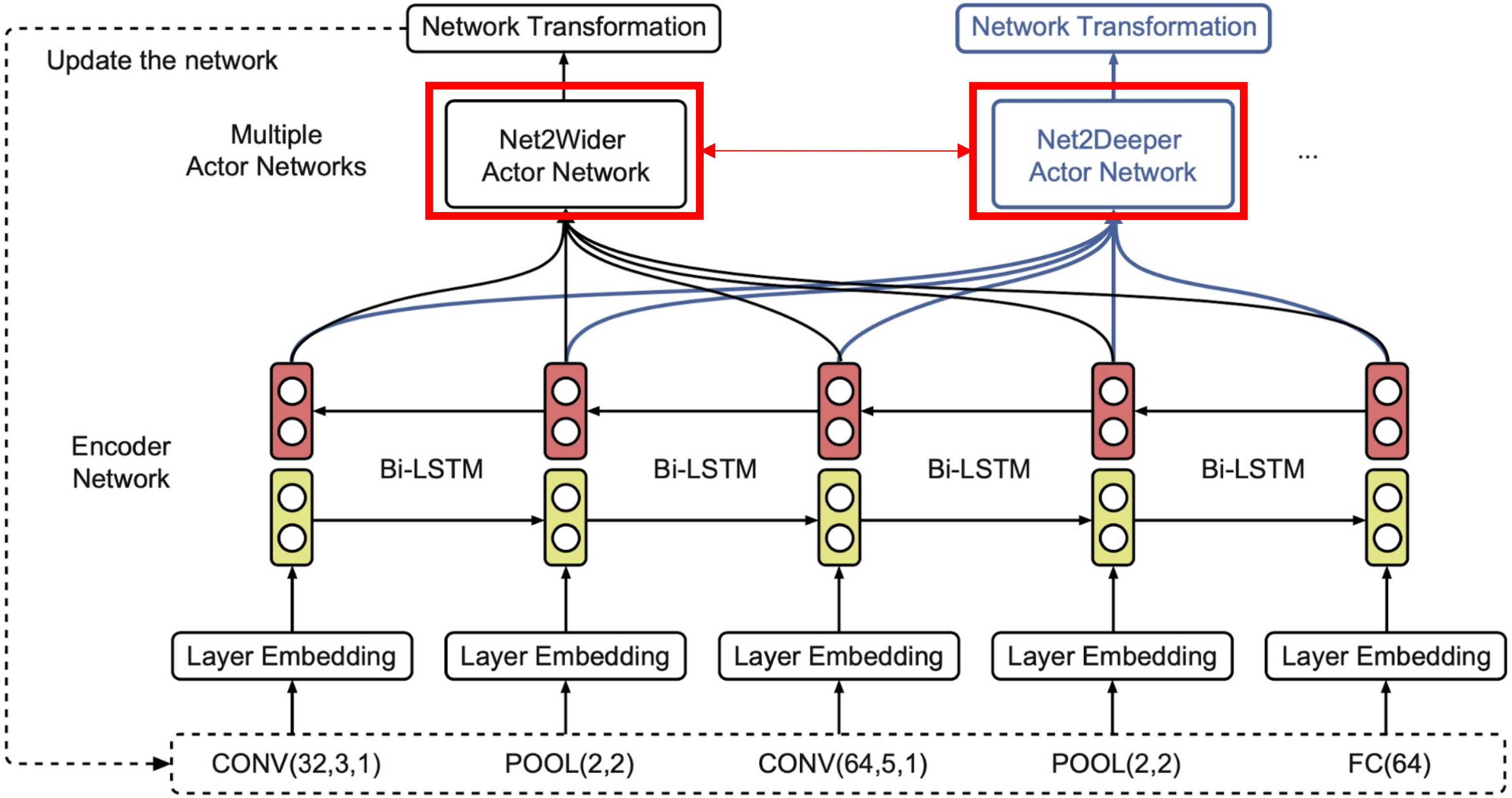


Cai, Han, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. "Efficient architecture search by network transformation." In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1. 2018.
https://arxiv.org/pdf/1707.04873.pdf

Parameter Sharing

more units for fully-connected layers
more filters for convolutional layers

insert a new layer
convolutional layer: the kernel is set to be identity filters
fully-connected layer: the weight matrix is set to be identity matrix



Cai, Han, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. "Efficient architecture search by network transformation." In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1. 2018.
https://arxiv.org/pdf/1707.04873.pdf

DARTS: differentiable architecture 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 weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
  2. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \mathrm{argmax}_{o \in \mathcal{O}}\, \alpha_o^{(i,j)}$ for each edge (i,j)

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

DARTS: differentiable architecture 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 weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
2. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \mathrm{argmax}_{o \in \mathcal{O}}\, \alpha_o^{(i,j)}$ for each edge (i,j)

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

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

Architecture

Corresponding weights

DARTS: differentiable architecture 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 weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
2. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \mathrm{argmax}_{o \in \mathcal{O}} \, \alpha_o^{(i,j)}$ for each edge (i,j)

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

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

Architecture

Corresponding weights

DARTS: differentiable architecture 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 weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
  2. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \mathrm{argmax}_{o \in \mathcal{O}} \, \alpha_o^{(i,j)}$ for each edge (i, j)

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

Architecture

Corresponding weights

DARTS: differentiable architecture 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 weights $\boxed{w \text{ by descending } \nabla_w \mathcal{L}_{train}(w, \alpha)}$
2. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}}\, \alpha_o^{(i,j)}$ for each edge (i,j)

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

Architecture

Corresponding weights

DARTS: differentiable architecture 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 weights $\boxed{w \text{ by descending } \nabla_w \mathcal{L}_{train}(w,\alpha)}$
2. Update architecture α by descending $\boxed{\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w,\alpha),\alpha)}$

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \mathrm{argmax}_{o \in \mathcal{O}}\, \alpha_o^{(i,j)}$ for each edge (i,j)

$$\min_\alpha \boxed{\mathcal{L}_{\text{validate}}(w^*(\alpha), \alpha)}$$

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

Architecture

Corresponding weights

DARTS: differentiable architecture search

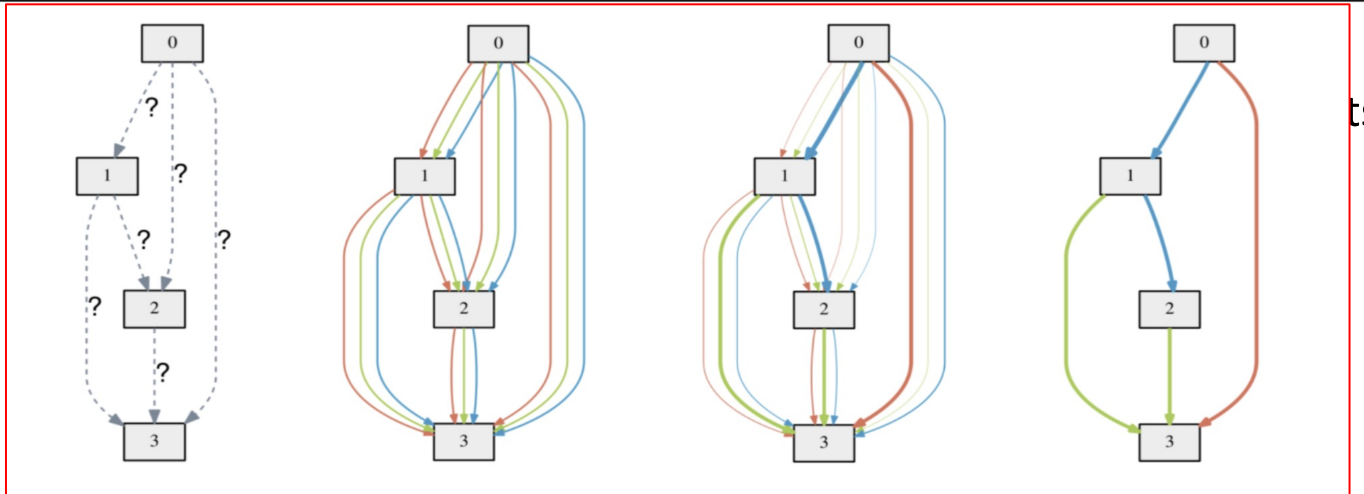
Algorithm 1: DARTS – Differentiable Architecture Search

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

while *not converged* **do**

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

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \, \alpha_o^{(i,j)}$ for each edge (i, j)



(a) Initially unknown operations on the edges.

(b) Continuous relaxation by placing a mixture of operations on each edge.

(c) Bilevel optimization to jointly train mixing probabilities and weights.

(d) Finalized the model based on the learned mixing probabilities.

DARTS: differentiable architecture search

Algorithm 1: DARTS – Differentiable Architecture Search

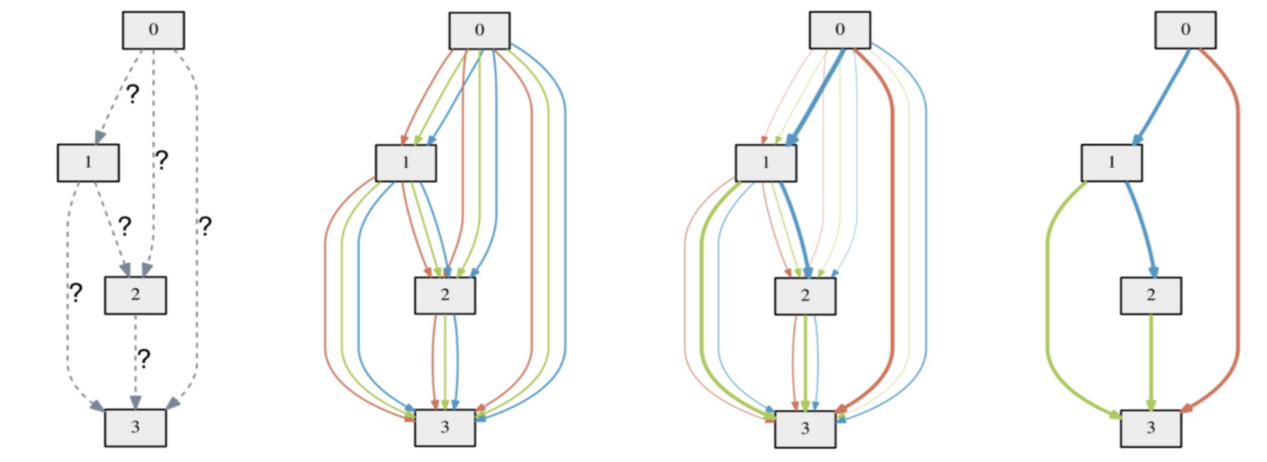
Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)
while *not converged* **do**
  1. Update weights $\boxed{w \text{ by descending } \nabla_w \mathcal{L}_{train}(w, \alpha)}$
  2. Update architecture α by descending $\boxed{\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)}$

Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \, \alpha_o^{(i,j)}$ for each edge (i, j)

$$x_i = \sum_{j<i} o^{(i,j)}(x_j)$$

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

ts



(a) Initially unknown operations on the edges.

(b) Continuous relaxation by placing a mixture of operations on each edge.

(c) Bilevel optimization to jointly train mixing probabilities and weights.

(d) Finalized the model based on the learned mixing probabilities.

DARTS: differentiable architecture 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 weights $\boxed{w \text{ by descending } \nabla_w \mathcal{L}_{train}(w, \alpha)}$
2. Update architecture α by descending $\boxed{\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)}$

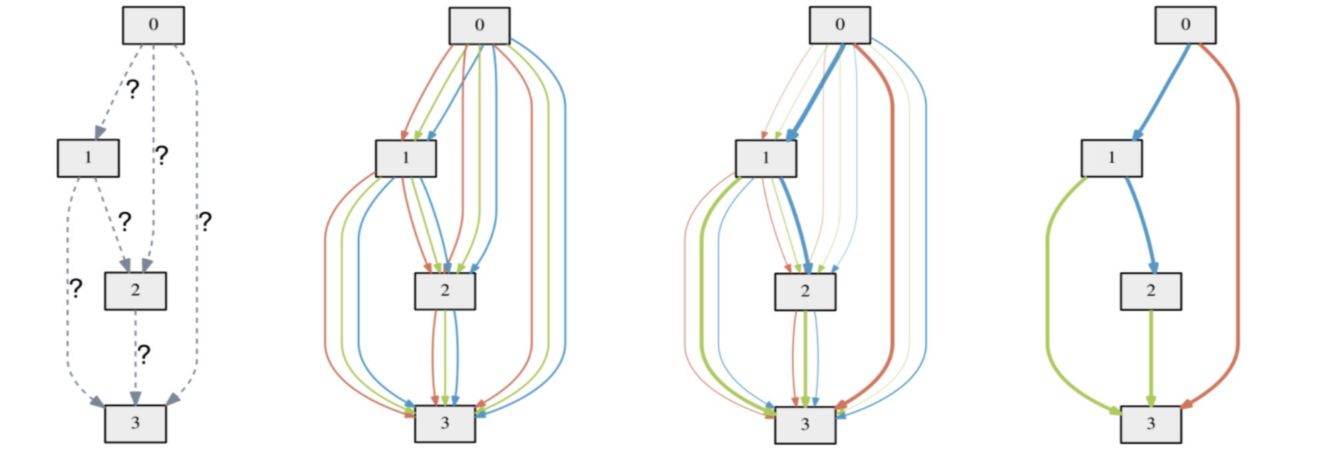
Replace $\bar{o}^{(i,j)}$ with $o^{(i,j)} = \mathrm{argmax}_{o \in \mathcal{O}}\, \alpha_o^{(i,j)}$ for each edge (i, j)

$$x_i = \sum_{j<i} o^{(i,j)}(x_j)$$

Mix of operations

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

differentiable



(a) Initially unknown operations on the edges.

(b) Continuous relaxation by placing a mixture of operations on each edge.

(c) Bilevel optimization to jointly train mixing probabilities and weights.

(d) Finalized the model based on the learned mixing probabilities.

ts

DARTS: differentiable architecture 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 weights $\boxed{w \text{ by descending } \nabla_w \mathcal{L}_{train}(w, \alpha)}$
  2. Update architecture α by descending $\boxed{\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)}$

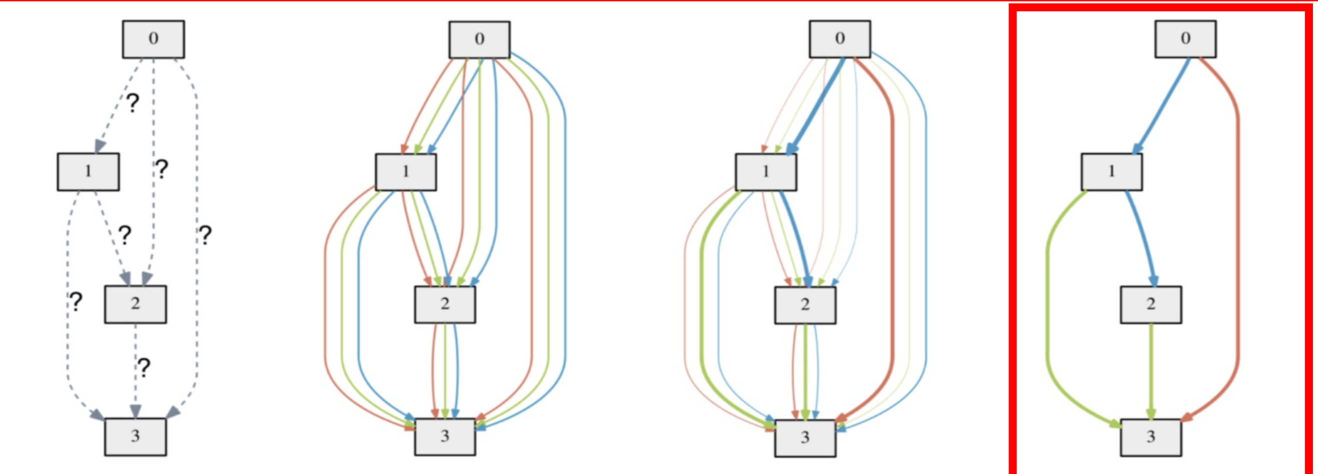
Replace $\bar{o}^{(i,j)}$ with $\boxed{o^{(i,j)} = \mathrm{argmax}_{o \in \mathcal{O}}\ \alpha_o^{(i,j)}}$ for each edge (i, j)

$$x_i = \sum_{j<i} o^{(i,j)}(x_j)$$

Mix of operations

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

differentiable



(a) Initially unknown operations on the edges.

(b) Continuous relaxation by placing a mixture of operations on each edge.

(c) Bilevel optimization to jointly train mixing probabilities and weights.

(d) Finalized the model based on the learned mixing probabilities.

ts