Journal meeting

- Efficient Neural Architecture Search (ENAS)

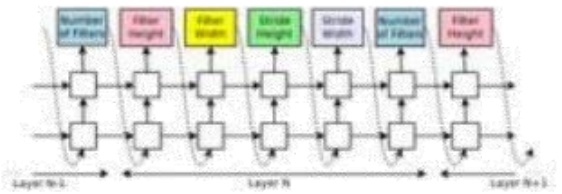
2020. 04. 17.

Hyunsoo, Yu
Heesang, Eom

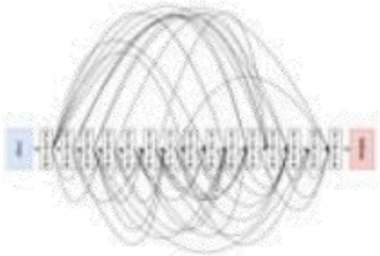
INDEX

- **NAS Review**
  - Procedure
    - *Generate model with RNN (controller)*
    - *Update controller parameter*
  - Limitations
- **ENAS**
  - Introduction
  - Methods
    - Designing Recurrent Cells
    - Designing Convolutionl Neural Networks
    - Training ENAS and Deriving Architectures
      - Training W (shared parameter of child models)
      - Training θ (parameter of controller)
  - Results
  - Conclusion

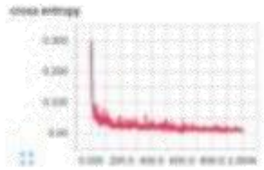
NAS Review - Procedure



1 Controller RNN is generate Neural Networks structure hyperparameters



2 Train is proceed after generate one child network from RNN controller



3 Record child network accuracy

$$\theta_c$$

θ_c: *RNN controllers parameter*

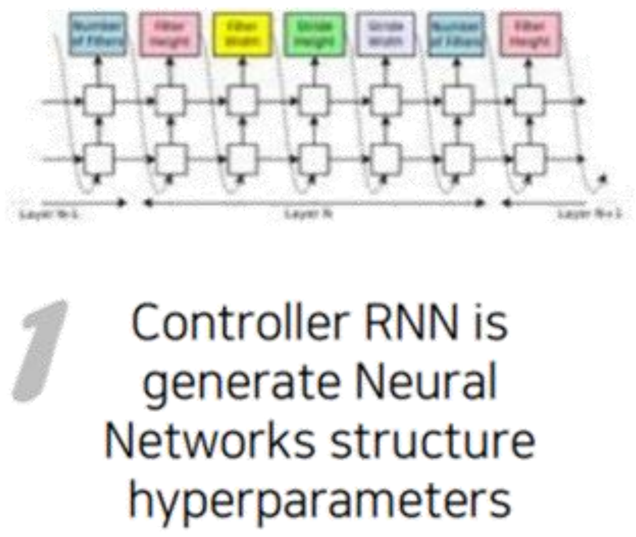
4 For Maximize Expected Validation Accuracy, θ_c is optimized

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

5 Update θ_c by using Policy gradient

NAS Review - Procedure

- **Generate model with RNN (Controller)**

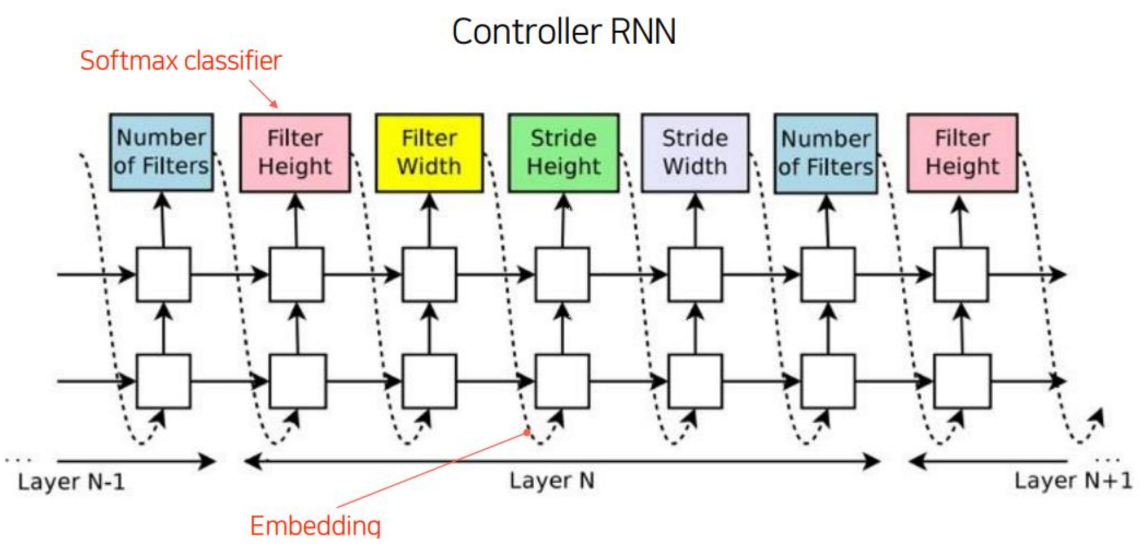


- We use RNN as controller.
  - For setting layer structure, string is used. (Tokenization)

```
["Filter Width: 5", "Filter Height: 3", "Num Filters: 24"]
```

  - Dealing with string, RNN is normally applied.

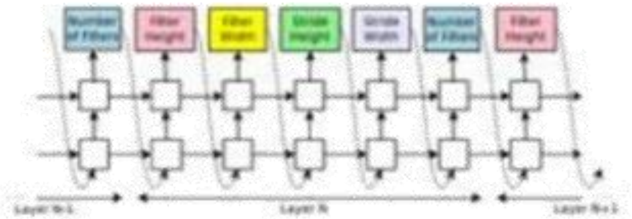
: Reason why RNN is used as controller



NAS Review - Procedure

Trainable parameter

- **Increase architecture complexity with 'skip connection'**



1 Controller RNN is generate Neural Networks structure hyperparameters

- Add anchor point

$$P(\text{Layer j is an input to layer i}) = \text{sigmoid}(v^{\mathrm{T}} \tanh(W_{prev} * h_j + W_{curr} * h_i))$$

- Solve 'Compilation failure' with
  - Different size => zero padding
  - Many input => concatenate
  - No input connection to layer => input : the image (First input)
  - In final layer => take all output

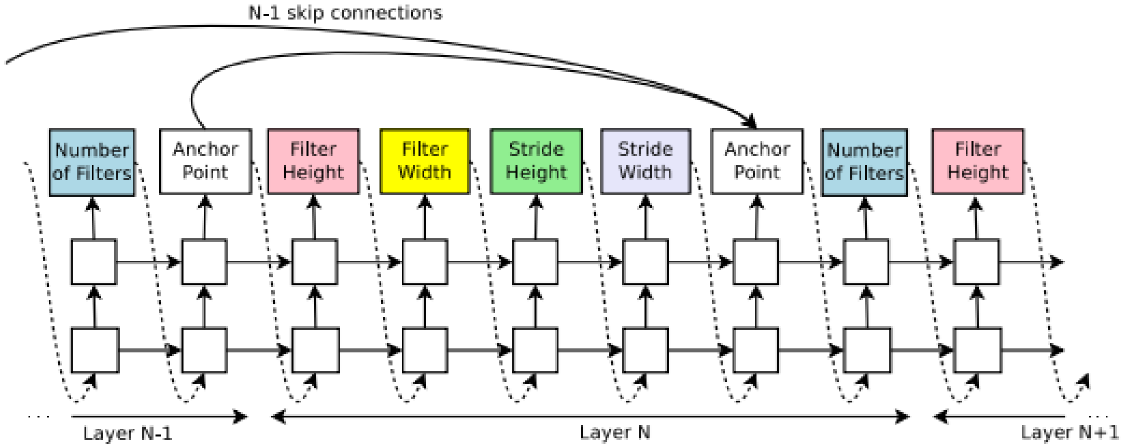


Figure 4: The controller uses anchor points, and set-selection attention to form skip connections.

NAS Review - Procedure

- **Update controller parameter θ**

5

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Update θ_c by using Policy gradient

* Method : Monte-Carlo Policy Gradient (REINFORCE)

: Because REINFORCE is easy to tuning

Number of hyperparameters that controller has to predict for design new neural network structure

Controller RNN parameters

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)R_k$$

Number of models in minibatch

Base line for reduce high variance in this prediction

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)(R_k - b)$$

NAS Review - Procedure

- **Update controller parameter θ**

5

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Update θ_c by using Policy gradient

* each gradient update to the controller parameters θ

= **train one child network to convergence**

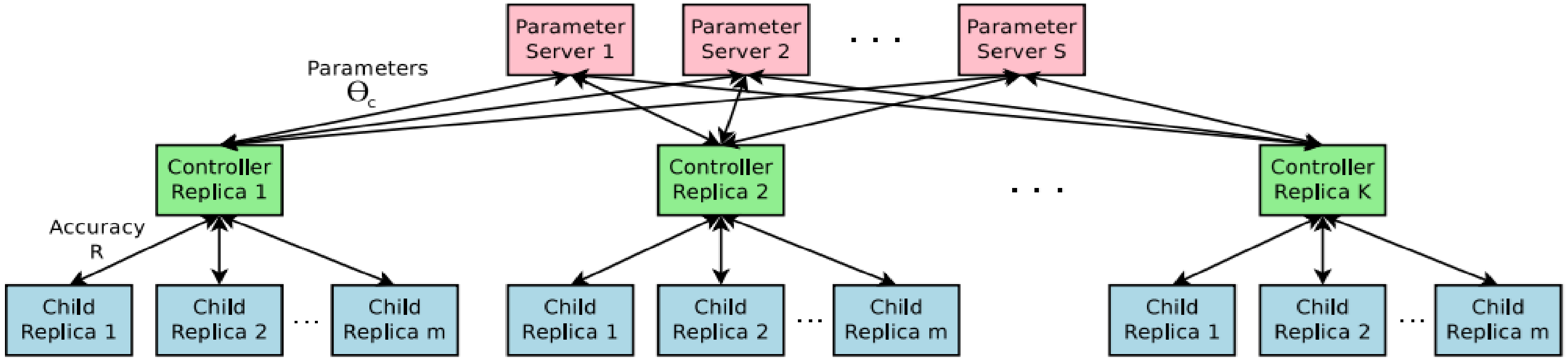
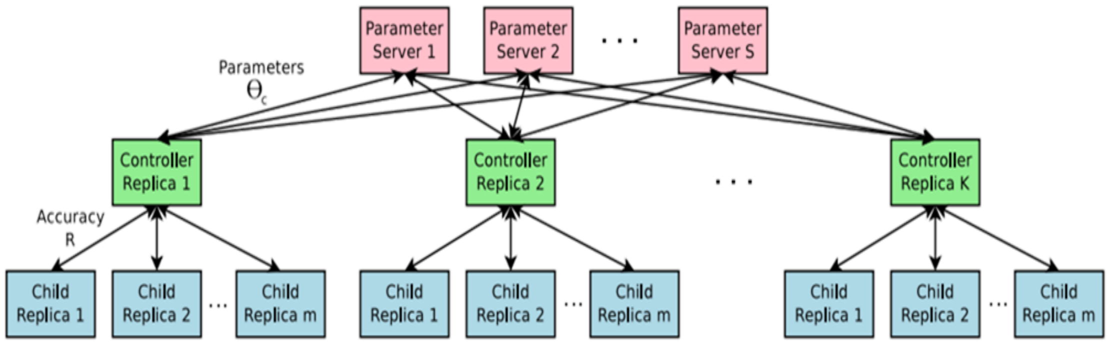


Figure 3: Distributed training for Neural Architecture Search. We use a set of S parameter servers to store and send parameters to K controller replicas. Each controller replica then samples m architectures and run the multiple child models in parallel. The accuracy of each child model is recorded to compute the gradients with respect to θ_c, which are then sent back to the parameter servers.

NAS Review - Limitations



<Parameter-server scheme to speed up learning process >

- Despite of this scheme, it takes too long.
  - take 'one month' with 800 GPUs in CIFAR-10 image classification
- **Cause** : child models don't reuse weights.
  - After making new model, they throw away weights and learn again.

Solution : ENAS

Efficient Neural Architecture Search

ENAS - Introduction

- NAS is computationally expensive and time consuming, *e.g.* Zoph et al.(2018) use 450 GPUs for 3-4 days (*i.e.* 32,400-43,200 GPU hours).
  - ENAS : using single GTX 1080Ti GPU, the search for architectures takes less than 16hours (compared to NAS, reduction is more than 1000x)

- Authors observe that the computational bottleneck of NAS is the training of each child model to convergence, only to measure its accuracy whilst throwing away all the trained weights.

- Improving the efficiency of NAS by forcing all child models to share weights to eschew training each child model from scratch to convergence.

ENAS - Methods

- The main idea of ENAS is the observation that <u>all of the graphs which NAS ends up iterating over can be viewed as sub-graphs of a larger graph</u>.
  -> NAS's search space can be represented as a single directed acyclic graph (**DAG**)

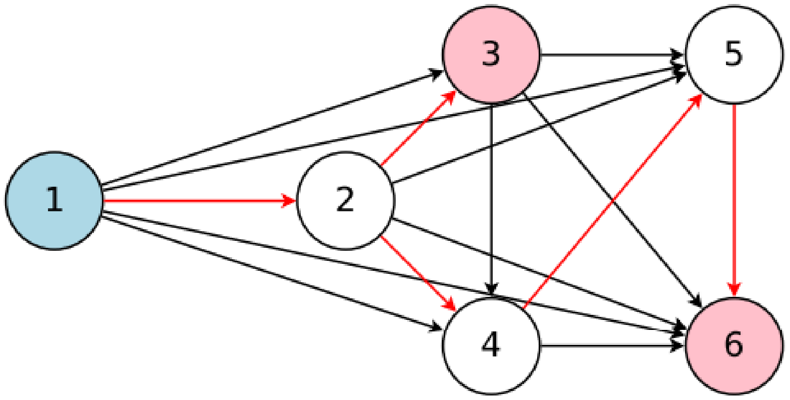


Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

- Whole line
  - Entire model which have chance to be selected
- Red line
  - Selected model (optimal model)

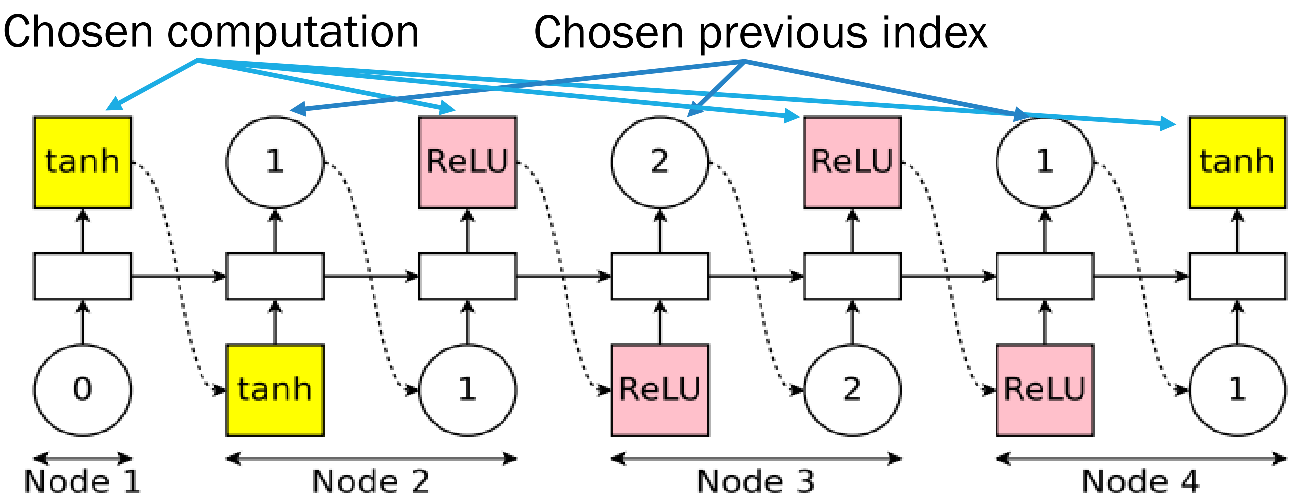
ENAS - Methods

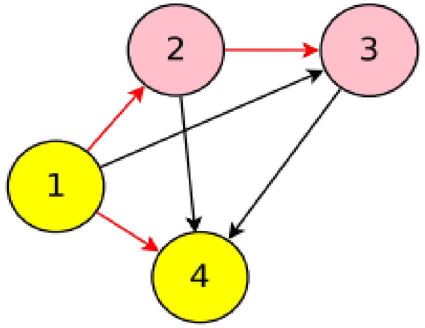
- **Designing Recurrent Cells**

- To design recurrent cells, a DAG with N nodes are selected, where the nodes represent local computations, and the edges represent the flow of information between the N nodes.

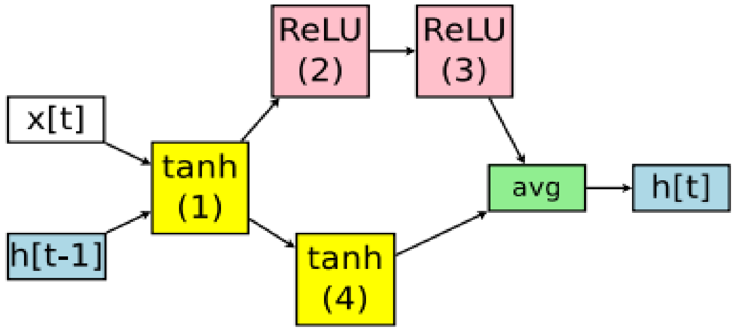
- ENAS's controller is an RNN that decides:
  - edges are activated and computations are performed at each node in the DAG

ENAS - Methods





DAG
(Directed Acyclic Graph)



Chosen model

: Controller(RNN)

- The role of Controller(RNN)
  - Choose previous index(j)
  - Choose which computation to do

$$h_1 = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}^{(\mathbf{x})} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(\mathbf{h})}\right)$$

$$h_2 = \mathrm{ReLU}(h_1 \cdot \mathbf{W}_{2,1}^{(\mathbf{h})})$$

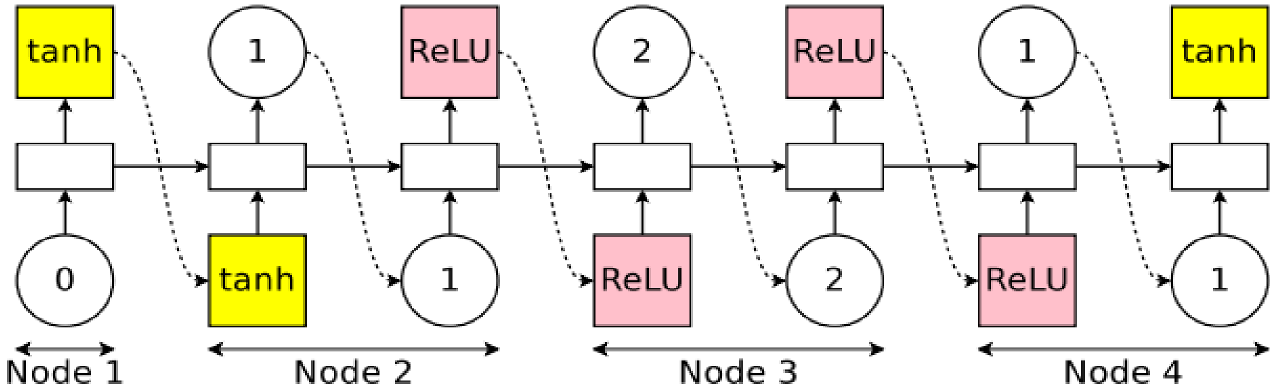
$$h_3 = \mathrm{ReLU}(h_2 \cdot \mathbf{W}_{3,2}^{(\mathbf{h})})$$

$$h_4 = \tanh\left(h_1 \cdot \mathbf{W}_{4,1}^{(\mathbf{h})}\right)$$

$$h_t = (h_3 + h_4)/2$$

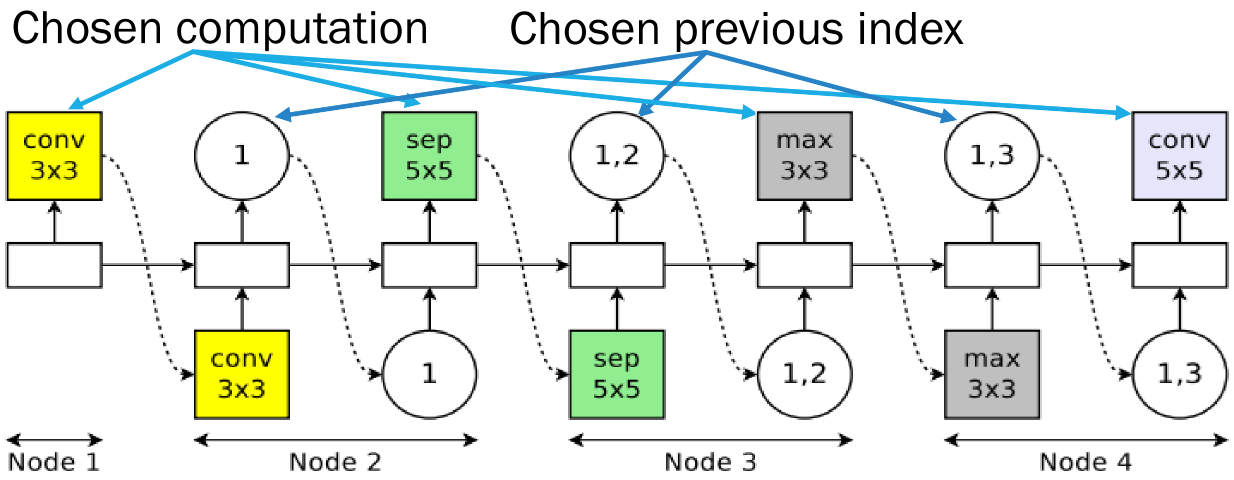
ENAS - Methods

- There is an independent parameter matrix $\mathbf{W}^{(\mathbf{h})}_{\ell,j}$ (each pair of nodes)
- By choosing the previous indices, the controller also decides which parameter matrices are used.
- Therefore, in ENAS, all recurrent cells in a search space **share the same set of parameters**.



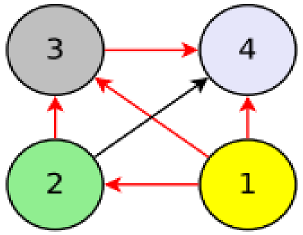
ENAS - Methods

- **Designing Convolutionl Neural Network**

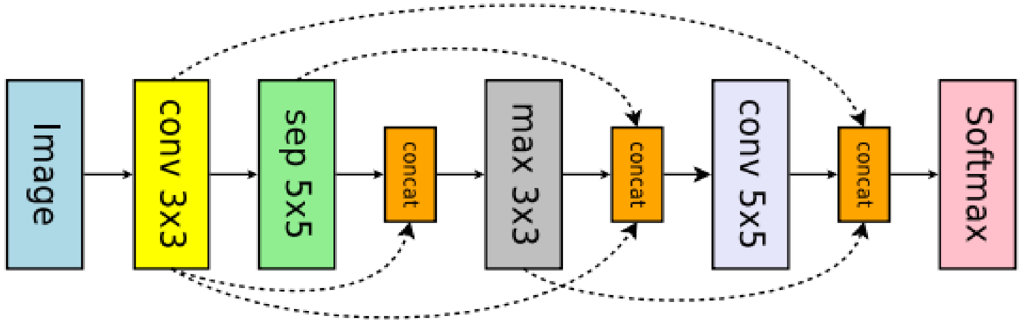


: Controller(RNN)

- The role of Controller(RNN)
  - Choose previous index(j)
  - Choose which computation to do



DAG
(Directed Acyclic Graph)



Chosen model

ENAS - Methods

- Training ENAS and Deriving Architectures
  - In ENAS, there are two sets of learnable parameters

- Training the shared parameters ω of the child models
- Training the controller parameters θ

- The training procedure of ENAS consists of two interleaving phases

ENAS - Methods

- Training the shared parameters ω

- Fix the controller's policy $\pi(\mathbf{m};\theta)$ and perform stochastic gradient descent (SGD) on ω to minimize the expected loss function

$$\mathbb{E}_{\mathbf{m}\sim\pi}\left[\mathcal{L}(\mathbf{m};\omega)\right].$$

- $\mathcal{L}(\mathrm{m};\omega)$: standard cross-entropy loss, computed on a training data, with a model m sampled from $\pi(\mathrm{m};\theta)$

$$\underbrace{\nabla_{\omega}\mathbb{E}_{\mathbf{m}\sim\pi(\mathbf{m};\theta)}\left[\mathcal{L}(\mathbf{m};\omega)\right]}_{\text{Unbiased estimation}} \approx \frac{1}{M}\sum_{i=1}^{M}\nabla_{\omega}\mathcal{L}(\mathbf{m}_i,\omega)$$

ENAS - Methods

- Training the controller parameters θ

- fix ω and update the policy parameters θ, aiming to maximize the expected reward $\mathbb{E}_{\mathbf{m}\sim\pi(\mathbf{m};\theta)}\left[\mathcal{R}(\mathbf{m},\omega)\right]$

- the gradient is computed using REINFORCE, with a moving average baseline to reduce variance.

- The reward $\mathcal{R}(\mathbf{m},\omega)$ is computed on the validation set (to avoid overfitting)

ENAS - Results

- **PPL** : Perplexity

Architecture	Additional Techniques	Params (million)	Test PPL
LSTM (Zaremba et al., 2014)	Vanilla Dropout	66	78.4
LSTM (Gal & Ghahramani, 2016)	VD	66	75.2
LSTM (Inan et al., 2017)	VD, WT	51	68.5
LSTM (Melis et al., 2017)	Hyper-parameters Search	24	59.5
LSTM (Yang et al., 2018)	VD, WT, ℓ_2, AWD, MoC	22	57.6
LSTM (Merity et al., 2017)	VD, WT, ℓ_2, AWD	24	57.3
LSTM (Yang et al., 2018)	VD, WT, ℓ_2, AWD, MoS	**22**	**56.0**
RHN (Zilly et al., 2017)	VD, WT	24	66.0
NAS (Zoph & Le, 2017)	VD, WT	54	62.4
ENAS	VD, WT, ℓ_2	**24**	**55.8**

Method	GPUs	Times (days)	Params (million)	Error (%)
DenseNet-BC (Huang et al., 2016)	–	–	25.6	3.46
DenseNet + Shake-Shake (Gastaldi, 2016)	–	–	26.2	2.86
DenseNet + CutOut (DeVries & Taylor, 2017)	–	–	26.2	**2.56**
Budgeted Super Nets (Veniat & Denoyer, 2017)	–	–	–	9.21
ConvFabrics (Saxena & Verbeek, 2016)	–	–	21.2	7.43
Macro NAS + Q-Learning (Baker et al., 2017a)	10	8-10	11.2	6.92
Net Transformation (Cai et al., 2018)	5	2	19.7	5.70
FractalNet (Larsson et al., 2017)	–	–	38.6	4.60
SMASH (Brock et al., 2018)	1	1.5	16.0	4.03
NAS (Zoph & Le, 2017)	800	21-28	7.1	4.47
NAS + more filters (Zoph & Le, 2017)	800	21-28	37.4	**3.65**
ENAS + macro search space	1	0.32	21.3	4.23
ENAS + macro search space + more channels	1	0.32	38.0	**3.87**
Hierarchical NAS (Liu et al., 2018)	200	1.5	61.3	3.63
Micro NAS + Q-Learning (Zhong et al., 2018)	32	3	–	3.60
Progressive NAS (Liu et al., 2017)	100	1.5	3.2	3.63
NASNet-A (Zoph et al., 2018)	450	3-4	3.3	3.41
NASNet-A + CutOut (Zoph et al., 2018)	450	3-4	3.3	**2.65**
ENAS + micro search space	1	0.45	4.6	3.54
ENAS + micro search space + CutOut	1	0.45	4.6	**2.89**

- **Macro search** : the controller designs the entire network.
- **Micro search** : the controller designs modules or building blocks, which are combined to build the final network

ENAS - Conclusion

- Conventional NAS's computational expense prevents it from being widely adopted.
- In this paper, ENAS, a novel method that speeds up NAS by more than 1000x, in terms of GPU hours.
- ENAS's key contribution is the sharing of parameters across child models during the search for architectures.
  - This insight is implemented by searching for a subgraph within a larger graph that incorporates architectures in a search space

References

- https://arxiv.org/pdf/1709.07
- https://jamiekang.github.io/2017/06/19/neural-arc
- https://s3.us-west-2.amazonaws.com/secure.notion-static.com/1ef8107c-3f52-4ec9-a988-95df0c91f877/NASBSW.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAT73L2G45O3KS52Y5%2F20200413%2Fus-west-2%2Fs3%2Faws4_request&X-Amz-Date=20200413T131137Z&X-Amz-Expires=86400&X-Amz-Signature=ada51171f02c7645cdca754685029917f3f1740b6674b791640ee0d19a30f956&X-Amz-SignedHeaders=host&response-content-disposition=filename%20%3D%22NAS%255BBSW%255D.pdf%22hitecture-search-with-reinforcement-learning/417.pdf
- https://arxiv.org/pdf/1802.03268.pdf

References

- http://www.secmem.org/blog/2019/07/19/Network-Architecture-Search/
- https://jamiekang.github.io/2017/06/19/neural-architecture-search-with-reinforcement-learning/
- https://www.slideshare.net/KihoSuh/neural-architecture-search-with-reinforcement-learning-76883153
- https://jayhey.github.io/deep%20learning/2018/03/15/ENAS/
- http://openresearch.ai/t/enas-efficient-neural-architecture-search-via-parameter-sharing/155
- https://www.slideshare.net/HoseongLee6/searching-for-activation-functions-paper-review