Neural Architecture Search

Ilya Trofimov
ilya.trofimov@skoltech.ru

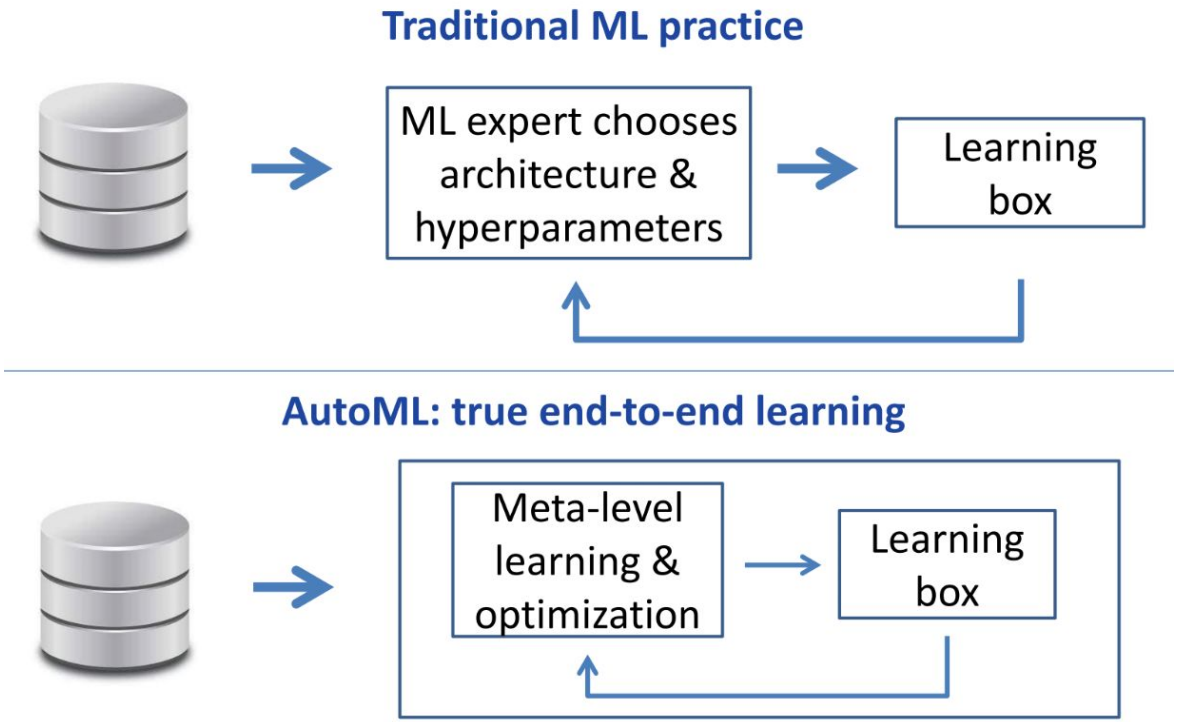
23.02.26

Neural Architecture Search

Outline:

1. Problem statement
2. Sample-based NAS
3. Differentiable NAS
4. NAS Benchmarks
5. Train-free NAS
6. Applications of NAS
7. NAS software

AutoML & Neural Architecture Search

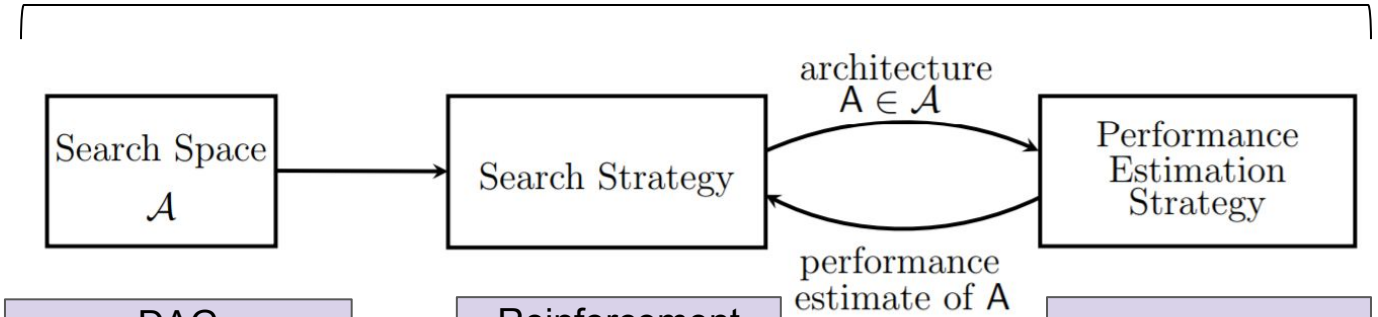


F. Hutter, (2020). AutoML Workshop, KDD.

General overview of Neural Architecture Search

Neural Architecture Search (NAS) is a part of AutoML field,
algorithm that ***searches*** for the best ***neural network architecture***.

Main Components of NAS



DAG representation	Reinforcement learning **(ENAS)**	Full training
Cell Block	Evolutionary Search	Partial training
Meta-Architecture	Gradient-Based Optimization(**DARTS, FBNet**)	Weight Sharing

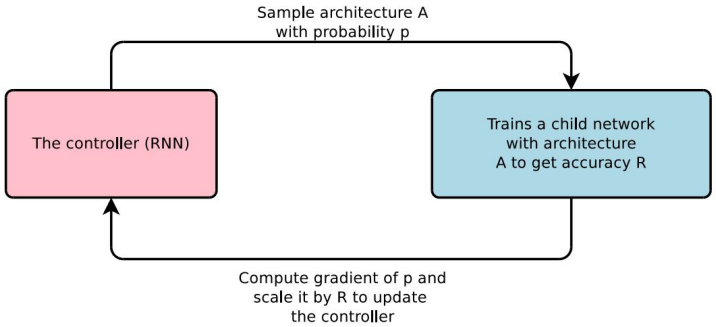
Neural Architecture Search typology

Family	Description
Sample based	train architectures from scratch
Weight-sharing	train a super-networks containing all the architectures as subgraphs
Train-free	attempts to predict network’s accuracy from random initialisation

Sample-based Neural Architecture Search

Giving birth to NAS

- The RL agent build network architecture layer by layer (SDP);
- Reward = validation accuracy after training;
- RL agent is trained with REINFORCE.



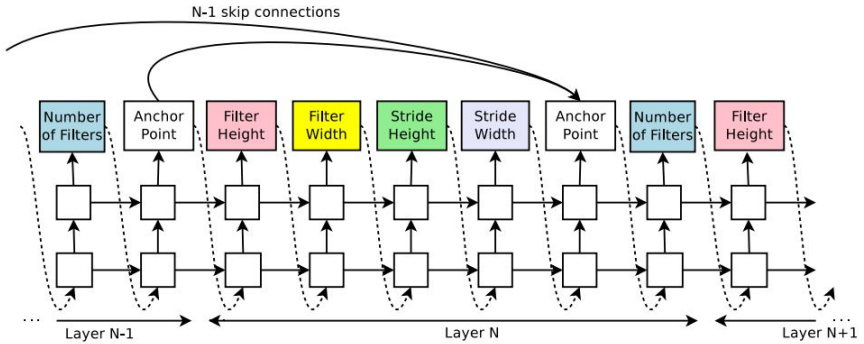


Figure 1: An overview of Neural Architecture Search.

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Results: After the controller trains 12,800 architectures, we find the architecture that achieves the best validation accuracy. We then run a small grid search over learning rate, weight decay, batchnorm epsilon and what epoch to decay the learning rate. The best model from this grid search is then run until convergence and we then compute the test accuracy of such model and summarize the results in Table 1. As can be seen from the table, Neural Architecture Search can design several promising architectures that perform as well as some of the best models on this dataset.

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

Table 1: Performance of Neural Architecture Search and other state-of-the-art models on CIFAR-10.

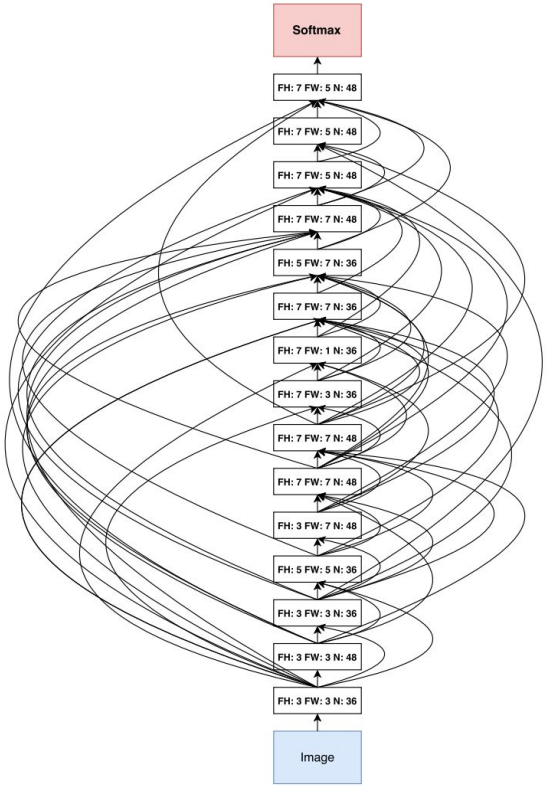


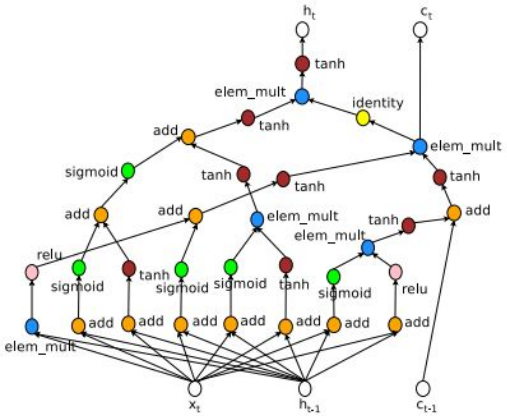
Figure 7: Convolutional architecture discovered by our method, when the search space does not have strides or pooling layers. FH is filter height, FW is filter width and N is number of filters. Note that the skip connections are not residual connections. If one layer has many input layers then all input layers are concatenated in the depth dimension.

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Giving birth to NAS

Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	79.7
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	78.6
Gal (2015) - Variational LSTM (large, untied)	66M	75.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	73.4
Kim et al. (2015) - CharCNN	19M	78.9
Press & Wolf (2016) - Variational LSTM, shared embeddings	51M	73.2
Merity et al. (2016) - Zoneout + Variational LSTM (medium)	20M	80.6
Merity et al. (2016) - Pointer Sentinel-LSTM (medium)	21M	70.9
Inan et al. (2016) - VD-LSTM + REAL (large)	51M	68.5
Zilly et al. (2016) - Variational RHN, shared embeddings	24M	66.0
Neural Architecture Search with base 8	32M	67.9
Neural Architecture Search with base 8 and shared embeddings	25M	64.0
Neural Architecture Search with base 8 and shared embeddings	54M	62.4

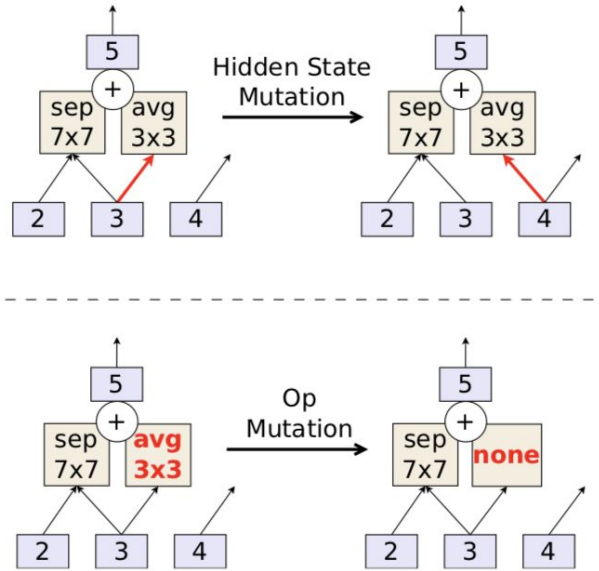
Table 2: Single model perplexity on the test set of the Penn Treebank language modeling task. Parameter numbers with ‡ are estimates with reference to Merity et al. (2016).



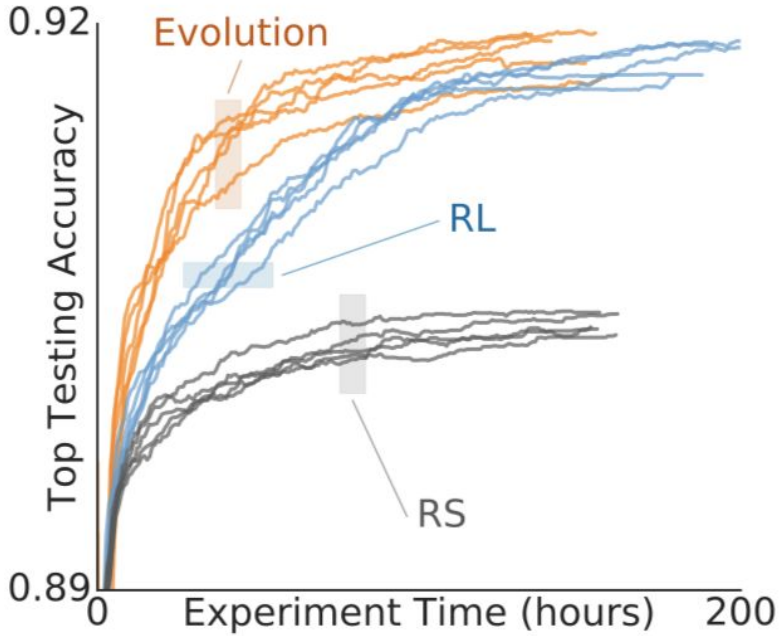
NAS Found RNN cell better than LSTM.

Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

AmoebaNET



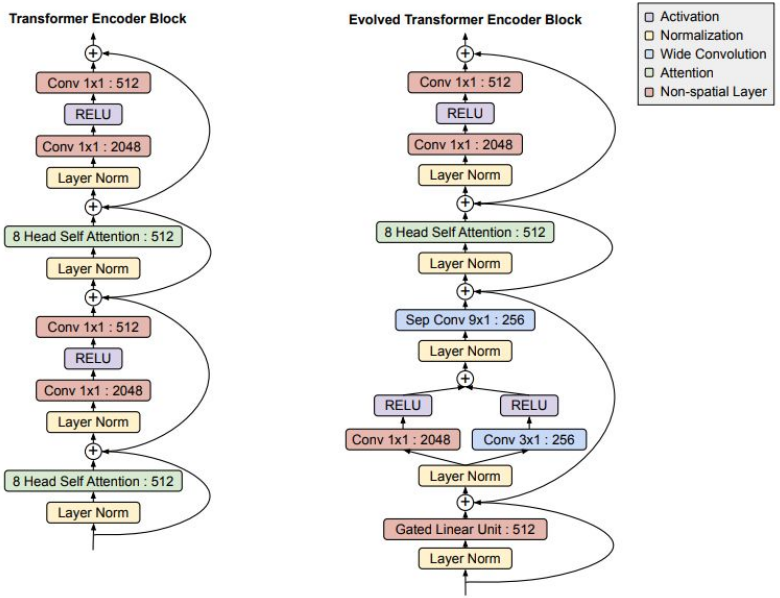
Evolutionary algorithm



RL-reinforcement learning, RS - random search

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019, July). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, No. 01, pp. 4780-4789).

The Evolved Transformer



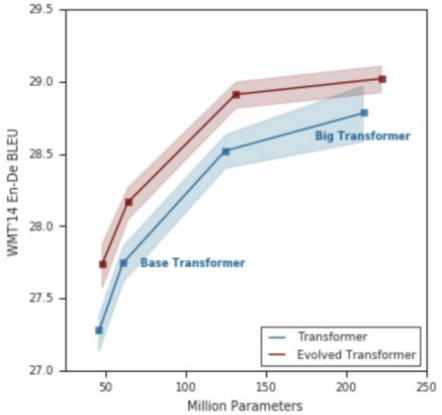


Figure 4. **Performance comparison of the Evolved Transformer against the Transformer across number of parameters.**

Search was performed with Progressive Dynamic Hurdles (PDH)

So, D., Le, Q., & Liang, C. (2019, May). The evolved transformer. In *International Conference on Machine Learning* (pp. 5877-5886). PMLR.

Neural Predictor via GNN

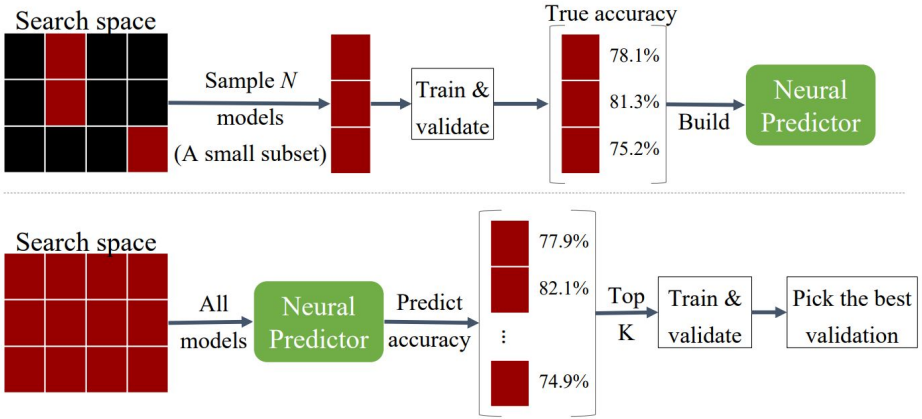


Figure 1: Training and applying the Neural Predictor.

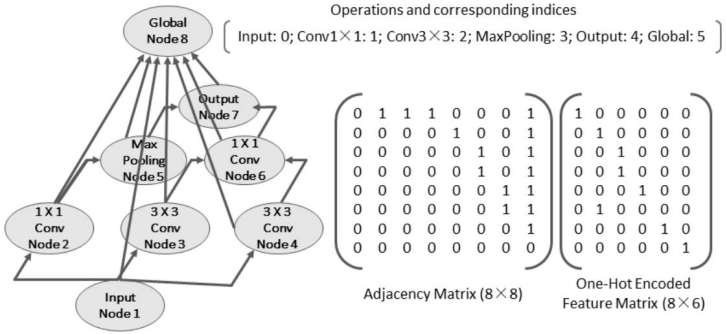
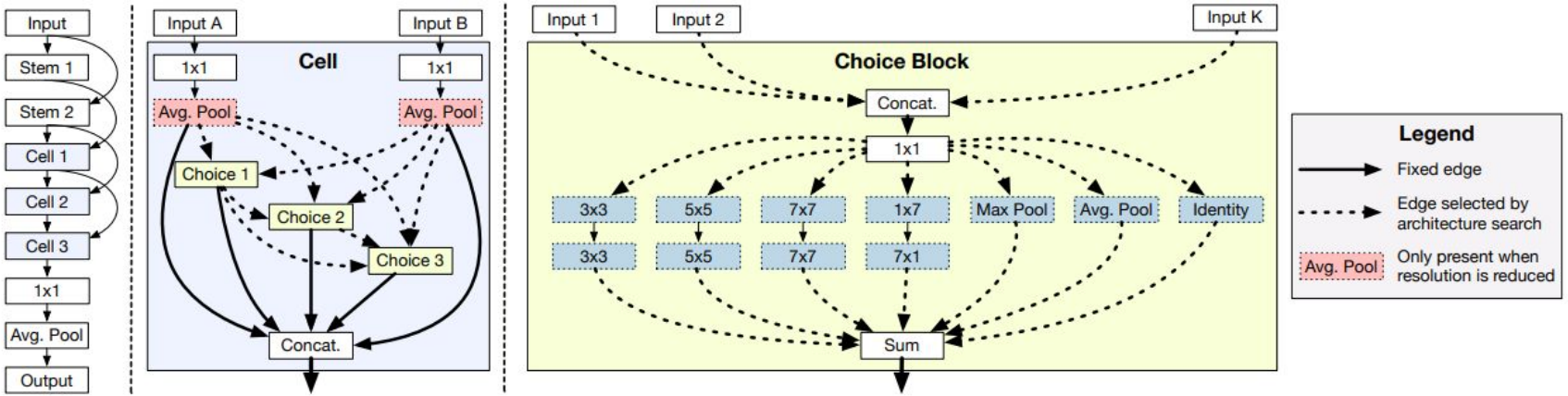


Figure 2: Encoding of an example cell in NAS-Bench-101.

Wen, W., Liu, H., Chen, Y., Li, H., Bender, G., & Kindermans, P. J. (2020). Neural predictor for neural architecture search. In European Conference on Computer Vision (pp. 660-676). Springer, Cham.

Weight-sharing trick



- All the networks from the search space share weights;
- Training resembles dropout.

Bender, G., Kindermans, P. J., Zoph, B., Vasudevan, V., & Le, Q. (2018, July). Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning* (pp. 550-559). PMLR.

Efficient Neural Architecture Search (ENAS)

ENAS = NAS + Weight Sharing

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
RHN (Zilly et al., 2017)	VD, WT	24	66.0
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**
NAS (Zoph & Le, 2017)	VD, WT	54	62.4
ENAS	VD, WT, ℓ_2	**24**	**56.3**

Table 1. Test perplexity on Penn Treebank of ENAS and other baselines. Abbreviations: RHN is *Recurrent Highway Network*, VD is *Variational Dropout*; WT is *Weight Tying*; ℓ_2 is *Weight Penalty*; AWD is *Averaged Weight Drop*; MoC is *Mixture of Contexts*; MoS is *Mixture of Softmaxes*.

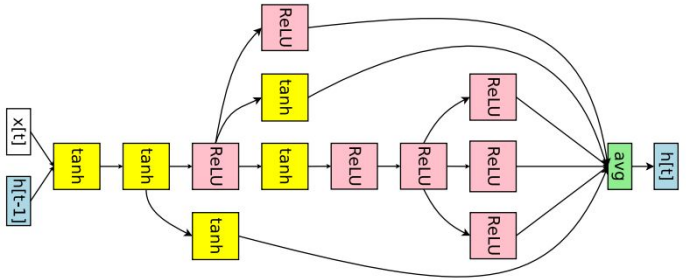


Figure 6. The RNN cell ENAS discovered for Penn Treebank.

ENAS is 1000x faster than NAS !

Pham, H., Guan, M., Zoph, B., Le, Q., & Dean, J. (2018, July). Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning* (pp. 4095-4104). PMLR.

DARTS (Differentiable ARchiTecture Search)

The space of candidate architectures is continuous, not discrete (compared to ENAS), which allows it to use gradient-based approaches.

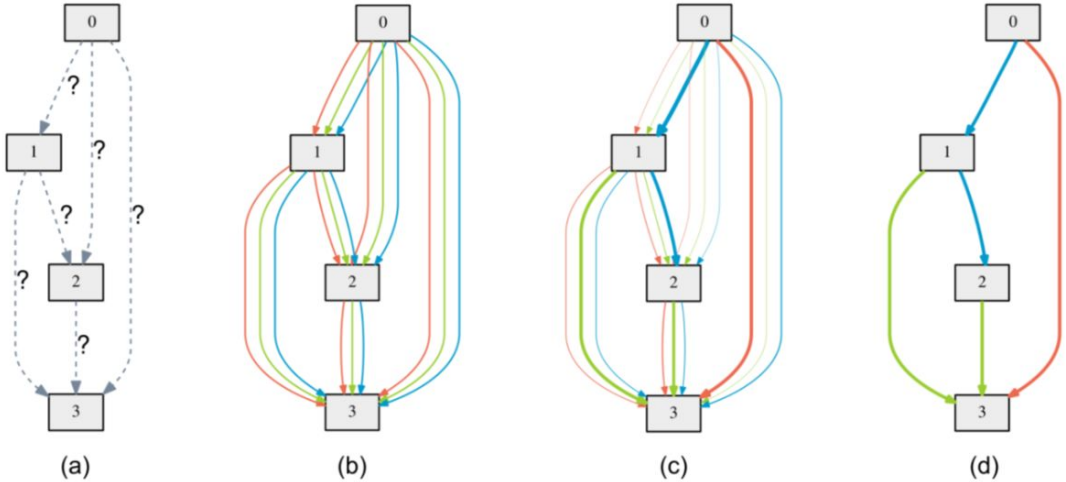


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

Operations on edges are mixed

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

8 operations:

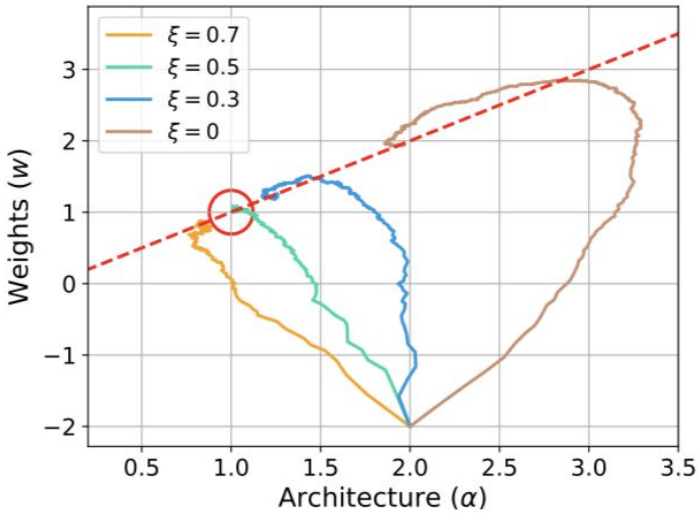
- 3x3, 5x5 sep. conv;
- 3x3, 5x5 dil. sep. conv;
- 3x3 max pooling
- 3x3 avg. pooling
- identity
- zero

Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

DARTS

Hyperparameter optimization is a bilevel optimization problem

$$\begin{aligned} \min_{\alpha} \quad & L_{val}(w^*(\alpha), \alpha) \\ s.t. \quad & w^*(\alpha) = \operatorname*{argmin}_{w} L_{train}(w, \alpha) \end{aligned}$$



Let's approximate $w^*(\alpha) \approx w' = w - \xi \nabla_w L_{train}(w, \alpha)$.
Then

$$\begin{aligned} \frac{d}{d\alpha} L_{val}(w^*(\alpha), \alpha) &\approx \nabla_\alpha L_{val}(w - \nabla_w L_{train}(w, \alpha), \alpha) \\ &= \nabla_w L_{val}(w', \alpha) \frac{\partial w'}{\partial \alpha} + \nabla_\alpha L_{val}(w', \alpha) \\ &= -\xi \nabla^2_{\alpha, w} L_{train}(w, \alpha) \nabla_w L_{val}(w', \alpha) + \nabla_\alpha L_{val}(w', \alpha) \end{aligned}$$

for matrix-vector product DARTS uses finite approximation

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

DARTS

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.

Modifications of DARTS

Many modifications of DARTS have been developed over time:

P-DARTS, PC-DARTS, DARTS+, DARTS-, GDAS, SDARTS, SGAS, DARTS+PT, DrNAS, etc.

Similar methods: FBNet, ProxylessNAS, SNAS, AutoDeepLab.

NAS Benchmarks

NAS Benchmarks

Issue 1: reproducibility.

Research papers use:

- Different training code;
- Different search spaces;
- Different evaluation schemas;

Issue 2: very time-consuming experiments.

Use the Same NAS Benchmarks, not Just the Same Datasets!

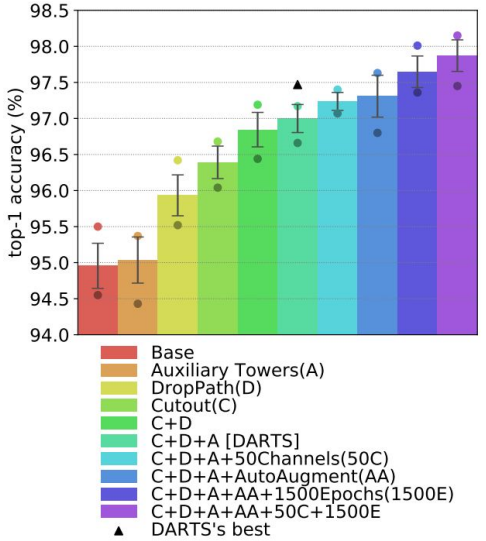


Figure 3: Comparison of different augmentation protocols for the DARTS search space on CIFAR10. Same colored dots represent minimum and maximum accuracies in the 8 runs.

Yang, A., Esperança, P. M., & Carlucci, F. M. (2019). Nas evaluation is frustratingly hard. *arXiv preprint arXiv:1912.12522*.

NAS-Bench-101

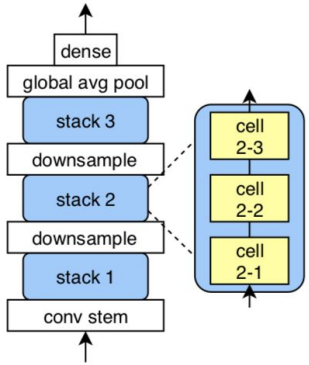
Tabular benchmark of trained and evaluated ~**423k** architectures of NN-s to compare multiple architectures on CIFAR-10 and compiled the results into a large dataset.

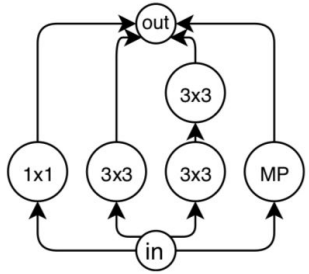
Cell configuration:

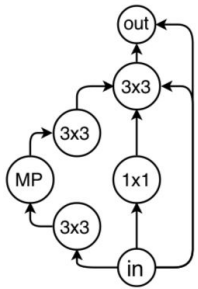
- Operations: 1x1 conv, 3x3 conv, max pool
- #Vertices ≤ 7
- #Edges ≤ 9

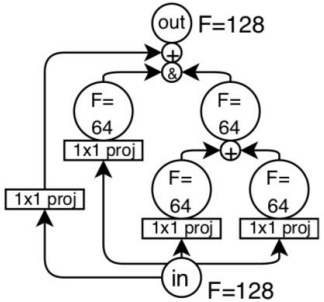
NAS-Bench-101 contains for all architectures:

- Validation accuracy after 12, 36, 108 epochs
- Testing accuracy after 12, 36, 108 epochs
- Training time



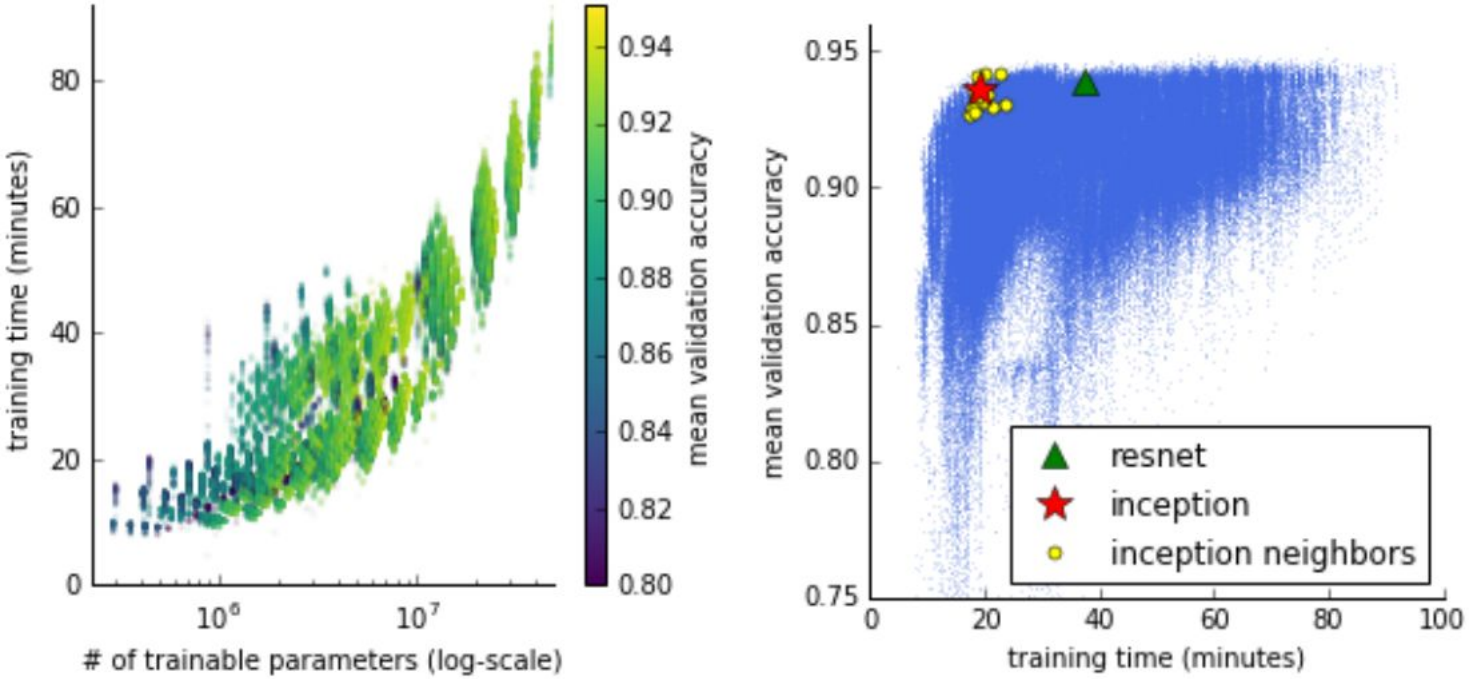






Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., & Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning* (pp. 7105-7114). PMLR.

NAS-Bench-101



NAS-Bench-101

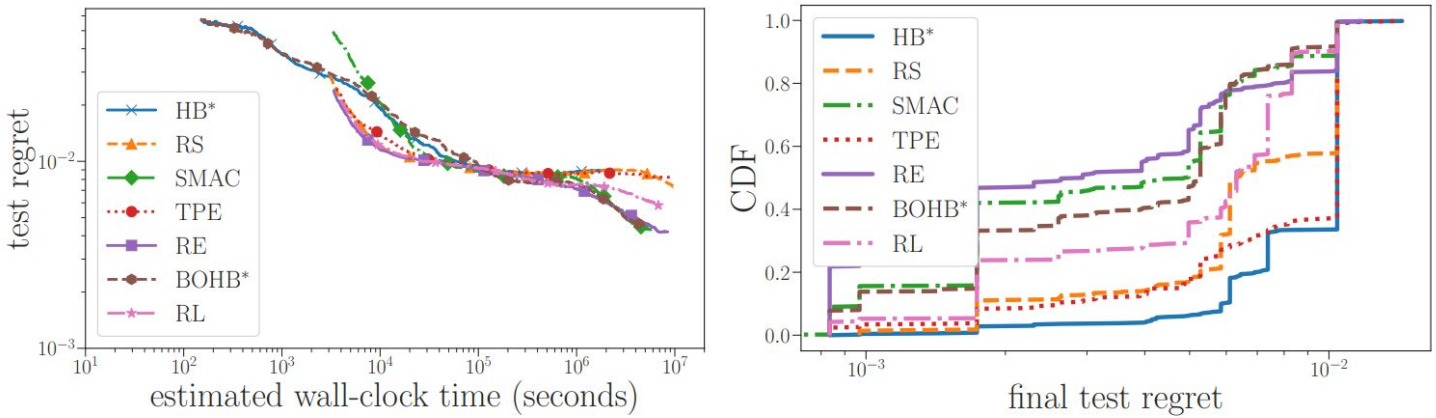


Figure 7: (left) Comparison of the performance of various search algorithms. The plot shows the mean performance of 500 independent runs as a function of the estimated training time. (right) Robustness of different optimization methods with respect to the seed for the random number generator. *HB and BO-HB are budget-aware algorithms which query the dataset a shorter epoch lengths. The remaining methods only query the dataset at the longest length (108 epochs).

Top methods: RE (regularized evolution), SMAC (bayes. opt), BOHB (multi-fidelity)

NAS-Bench-201

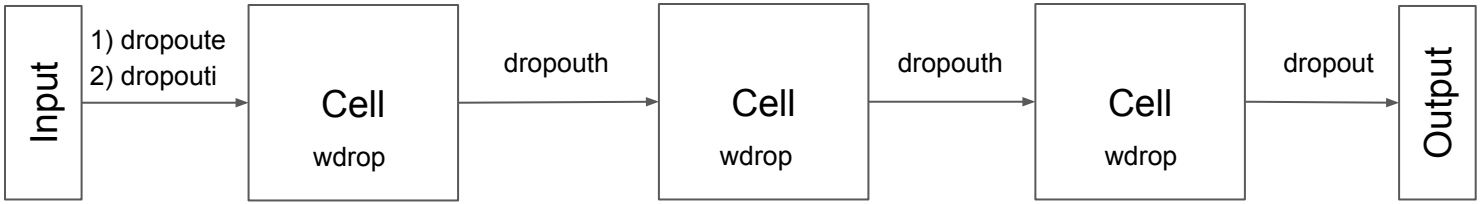
Method	Search (seconds)	CIFAR-10		CIFAR-100		ImageNet-16-120	
		validation	test	validation	test	validation	test
RSPS	8007.13	80.42±3.58	84.07±3.61	52.12±5.55	52.31±5.77	27.22±3.24	26.28±3.09
DARTS-V1	11625.77	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS-V2	35781.80	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS	31609.80	89.89±0.08	93.61±0.09	71.34±0.04	70.70±0.30	41.59±1.33	41.71±0.98
SETN	34139.53	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
ENAS	14058.80	37.51±3.19	53.89±0.58	13.37±2.35	13.96±2.33	15.06±1.95	14.84±2.10
RSPS†	7587.12	84.16±1.69	87.66±1.69	59.00±4.60	58.33±4.34	31.56±3.28	31.14±3.88
DARTS-V1†	10889.87	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS-V2†	29901.67	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS†	28925.91	90.00±0.21	93.51±0.13	71.14±0.27	70.61±0.26	41.70±1.26	41.84±0.90
SETN†	31009.81	82.25±5.17	86.19±4.63	56.86±7.59	56.87±7.77	32.54±3.63	31.90±4.07
ENAS†	13314.51	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
REA	12000	91.19±0.31	93.92±0.30	71.81±1.12	71.84±0.99	45.15±0.89	45.54±1.03
RS		90.93±0.36	93.70±0.36	70.93±1.09	71.04±1.07	44.45±1.10	44.57±1.25
REINFORCE		91.09±0.37	93.85±0.37	71.61±1.12	71.71±1.09	45.05±1.02	45.24±1.18
BOHB		90.82±0.53	93.61±0.52	70.74±1.29	70.85±1.28	44.26±1.36	44.42±1.49
ResNet	N/A	90.83	93.97	70.42	70.86	44.53	43.63
optimal		91.61	94.37	73.49	73.51	46.77	47.31

NN's are trained for 12 epochs

Dong, X., & Yang, Y. (2020). Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*.

NAS-Bench-NLP: Search Space

We took AWD-LSTM as a macro-structure of RNNs:
a reasonable trade-off between architecture complexity and performance.



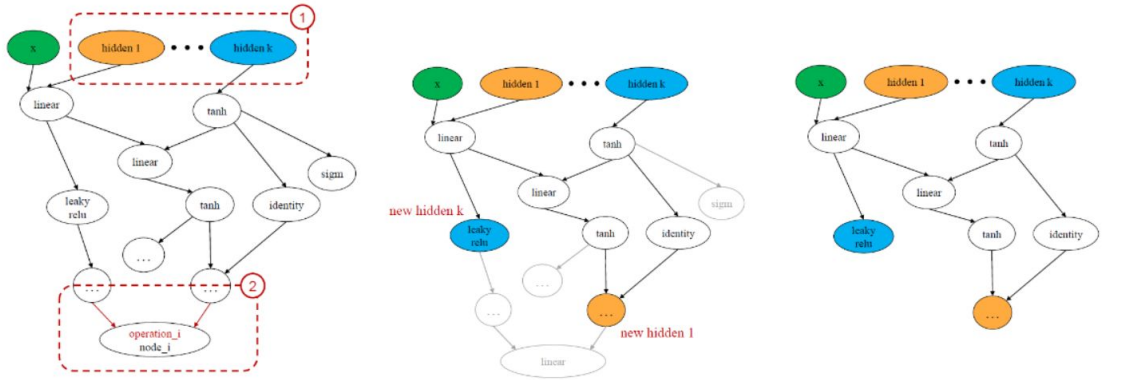
The dataset was PTB (1.086M tokens, vocabulary size = 1000)
https://github.com/fmsnew/nas-bench-nlp-release

Klyuchnikov, N., Trofimov, I., Artemova, E., Salnikov, M., Fedorov, M., & Burnaev, E. (2020). NAS-Bench-NLP: neural architecture search benchmark for natural language processing. *arXiv preprint arXiv:2006.07116*.

NAS-Bench-NLP: Search Space

Operations: linear layer, blending, elementwise_prod, elementwise_sum
Number of vertices ≤ 24
Number of edges < 100
Number of hidden states ≤ 3

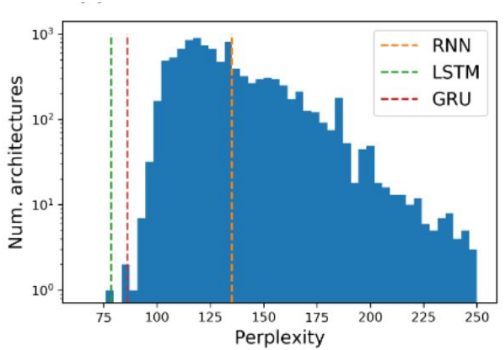
The search space includes RNN, LSTM, GRU by design.



(a) Stage 1 and 2: selecting the number of hidden states and adding nodes to the graph. (b) Stage 3: selecting new hidden nodes. (c) Stage 4: removing redundant nodes.

Figure 2: Illustrated stages of the architecture cell generation process.

NAS-Bench-NLP

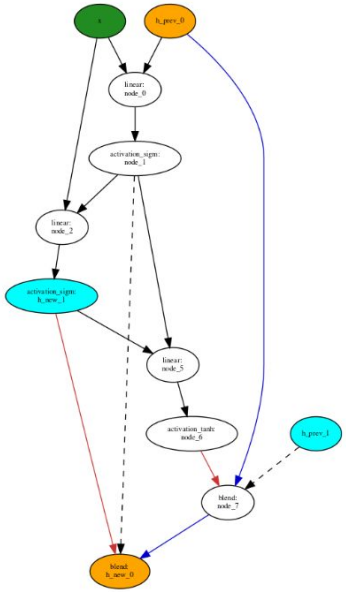


(b) Best test perplexity distribution.

Figure 4: Architectures metrics on PTB.

Architecture	Num. params	Test perplexity
LSTM	10.9M	78.5
Top-2	9.2M	84.7
GRU	9.2M	86.1
Top-4	11.0M	90.6
RNN	5.73M	135.1

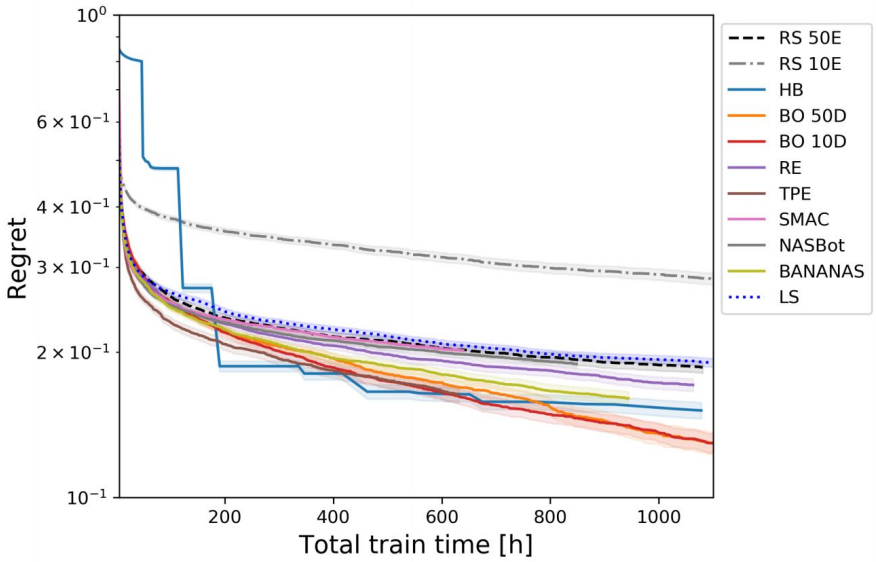
Table 3: Detailed comparison of top performing architectures and ordinary RNN on PTB dataset.



(a) Top-2 Cell

We have trained ~14k architectures from the search space.
The training took ~700 GPU*day with Zhores cluster.
We found a network (“top-2”) better than GRU.

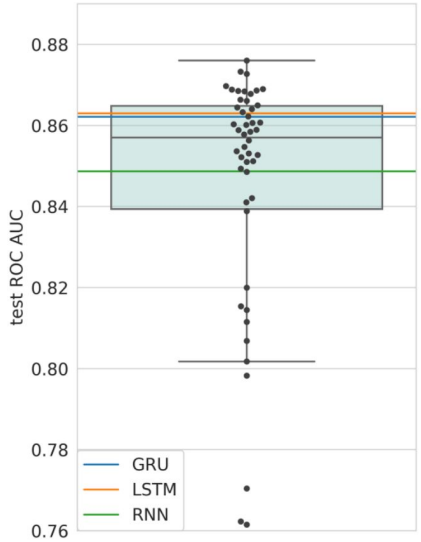
NAS-Bench-NLP



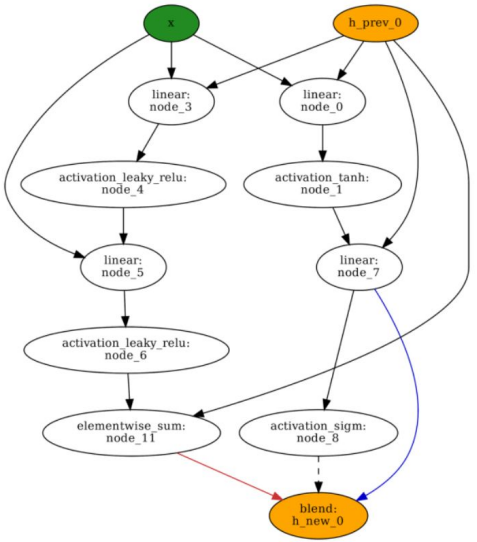
Bayesian optimization (via ensemble of XGBoost) is the best performing method.

NAS-Bench-NLP: Application to Bank Data

Evaluation of top-50 cells



The best cell has ROC AUC = 0.876



Performance of the architecture on the “Gender” dataset: gender classification from bank transaction data.

Train-free Neural Architecture Search

Train-free NAS (TE-NAS)

TE-NAS ranks architectures by analyzing the (1) spectrum of the neural tangent kernel (NTK) and the (2) number of linear regions in the input space.

$$\hat{\Theta}(\boldsymbol{x}, \boldsymbol{x}') = J(\boldsymbol{x})J(\boldsymbol{x}')^T, \text{ where } J_{i\alpha}(\boldsymbol{x}) = \partial_{\theta_\alpha} z_i^L(\boldsymbol{x})$$

z_i^L is the output of the i-th neuron in the last output layer L for parameter θ_α

- Calculate eigenvalues of NTK: $\lambda_0 \geq \cdots \geq \lambda_m$
- A metric of ***trainability***: $\kappa_{\mathcal{N}} = \frac{\lambda_0}{\lambda_m}$

Approximate calculation at batch size = 32

Chen, W., Gong, X., & Wang, Z. (2021). Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *ICLR' 2021.*

Train-free NAS (TE-NAS)

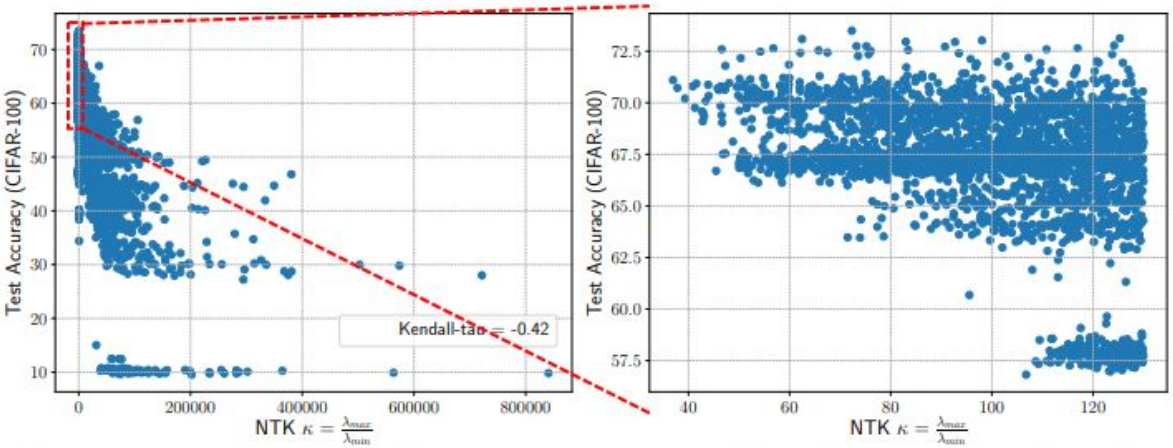


Figure 1: Condition number of NTK $\kappa_{\mathcal{N}}$ exhibits negative correlation with the test accuracy of architectures in NAS-Bench201 (Dong & Yang, 2020).

Train-free NAS (TE-NAS)

- ***Expressivity*** is defined as number of linear regions;
- ***Expressivity*** is calculated approximately by averaging three initializations and 5000 inputs;

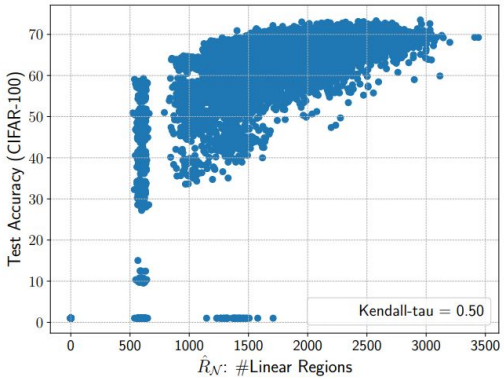


Figure 3: Number of linear regions $\hat{R}_{\mathcal{N}}$ of architectures in NAS-Bench201 exhibits positive correlation with test accuracies.

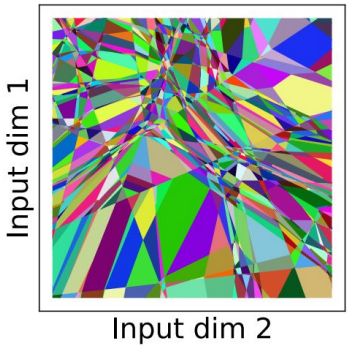


Figure 2: Example of linear regions divided by a ReLU network[1]

Train-free NAS (TE-NAS)

Algorithm 1: TE-NAS: Training-free Pruning-based NAS via Ranking of $\kappa_{\mathcal{N}}$ and $\hat{R}_{\mathcal{N}}$.

Input: supernet $\mathcal{N}_0$ stacked by cells, each cell has E edges, each edge has $|\mathcal{O}|$ operators, step $t = 0$.
while $\mathcal{N}_t$ *is not a single-path network* **do**
  for *each operator* o_j *in* $\mathcal{N}_t$ **do**
    $\Delta\kappa_{t,o_j} = \kappa_{\mathcal{N}_t} - \kappa_{\mathcal{N}_t \backslash o_j}$ ▷ the higher $\Delta\kappa_{t,o_j}$ the more likely we will prune o_j
    $\Delta R_{t,o_j} = R_{\mathcal{N}_t} - R_{\mathcal{N}_t \backslash o_j}$ ▷ the lower $\Delta R_{t,o_j}$ the more likely we will prune o_j
  Get importance by $\kappa_{\mathcal{N}}$: $s_\kappa(o_j) =$ index of o_j in descendingly sorted list $[\Delta\kappa_{t,o_1}, ..., \Delta\kappa_{t,o_{|\mathcal{N}_t|}}]$
  Get importance by $R_{\mathcal{N}}$: $s_R(o_j) =$ index of o_j in ascendingly sorted list $[\Delta R_{t,o_1}, ..., \Delta R_{t,o_{|\mathcal{N}_t|}}]$
  Get importance $s(o_j) = s_\kappa(o_j) + s_R(o_j)$
  $\mathcal{N}_{t+1} = \mathcal{N}_t$
  for *each edge* e_i, $i = 1, ..., E$ **do**
    $j^* = \arg\min_j \{s(o_j) : o_j \in e_i\}$ ▷ find the operator with greatest importance on each edge.
    $\mathcal{N}_{t+1} = \mathcal{N}_{t+1} \backslash o_{j^*}$
  $t = t + 1$
return *Pruned single-path network* $\mathcal{N}_t$.

Train-free NAS (TE-NAS)

Table 1: Comparison with state-of-the-art NAS methods on NAS-Bench-201. Test accuracy with mean and deviation are reported. "optimal" indicates the best test accuracy achievable in NAS-Bench-201 search space.

Architecture	**CIFAR-10**	**CIFAR-100**	**ImageNet-16-120**	**Search Cost (GPU sec.)**	**Search Method**
ResNet (He et al., 2016)	93.97	70.86	43.63	-	-
RSPS (Li & Talwalkar, 2020)	87.66(1.69)	58.33(4.34)	31.14(3.88)	8007.13	random
ENAS (Pham et al., 2018)	54.30(0.00)	15.61(0.00)	16.32(0.00)	13314.51	RL
DARTS (1st) (Liu et al., 2018b)	54.30(0.00)	15.61(0.00)	16.32(0.00)	10889.87	gradient
DARTS (2nd) (Liu et al., 2018b)	54.30(0.00)	15.61(0.00)	16.32(0.00)	29901.67	gradient
GDAS (Dong & Yang, 2019)	93.61(0.09)	70.70(0.30)	41.84(0.90)	28925.91	gradient
NAS w.o. Training (Mellor et al., 2020)	91.78(1.45)	67.05(2.89)	37.07(6.39)	4.8	training-free
TE-NAS (ours)	**93.9(0.47)**	**71.24(0.56)**	**42.38(0.46)**	1558	training-free
Optimal	94.37	73.51	47.31	-	-

Applications of Neural Architecture Search

Neural Architecture Search: Results

Problem	Dataset	NAS method	Human arc.perf.	NAS Perf.	Diff.
Machine translation	WMT'14 En-De	Evolution	BLEU=28.8 Perplexity=4.05 Transformer *[Vasnawi et al., 2017]*	BLEU=29.0 Perplexity=3.94 *Evolved Transformer* *[So et al. 2019]*	+0.7% -3.0% **lower is better**
Object classification	CIFAR-10	DARTS	Accuracy = 96.54% *DenseNet-BC, [Huang et al., 2017]*	Accuracy=97.24% *[Liu et al. 2018]*	+0.7%
Semantic segmentation	Cityscapes	Evolution	mIOU=71.8% *FRRN-B, [Pohlen et al., 2017]*	mIOU=80.4% *Auto-DeepLab-L* *[Liu et al, 2019]*	+8.6%

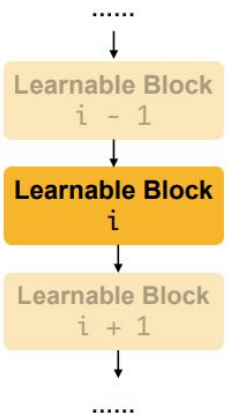
Neural Architecture Search: Results

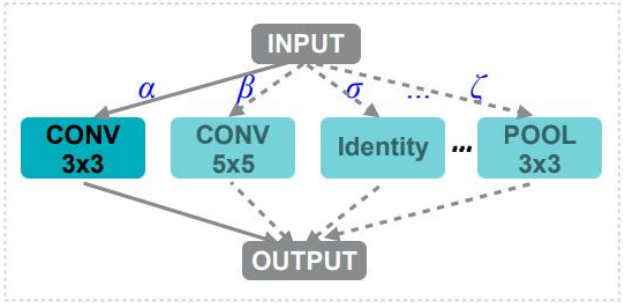
Problem	Dataset	NAS method	Human arc.perf.	NAS Perf.	Diff.
Natural language modeling	Penn Tree Bank	ENAS	Perplexity=56.0 *[Yang et al, 2018]*	Perplexity=55.8 *[Zoph et al, 2018]*	-0.3% **lower is better**
Graph NN, Classification	Citeseer	ENAS	Accuracy=73.0% *LGCN* *[Gao et al, 2018]*	Accuracy=73.8% *Auto-GNN* *[Zhou et al., 2019]*	+1%
Deep RL	Atari	ENAS	Avg. reward = 172.8 *NatureCNN* *[Mnih et al., 2015]*	Avg. reward = 181.8 *Skoltech, 2019*	+5.2%
Object detection	CoCo	DARTS	Avg.precision = 0.064 *[Law et al., 2018]*	Avg. precision=0.078 *Skoltech, 2019*	+1.4%

ImageNet benchmark / paperswithcode.com

RANK	MODEL	ACCURACY ↑	PARAMS	EXTRA TRAINING DATA	PAPER	CODE	RESULT	YEAR
1	CAIT-M36-448	90.2%		×	Going deeper with Image Transformers			2021
2	FixEfficientNet-B8	90.0%	87M	×	Fixing the train-test resolution discrepancy: FixEfficientNet			2020
3	DeiT-B-384	89.3%	86M	×	Training data-efficient image transformers & distillation through attention			2020
4	DeiT-B	88.7%	86M	×	Training data-efficient image transformers & distillation through attention			2020
5	Assemble-ResNet152	88.65%		×	Compounding the Performance Improvements of Assembled Techniques in a Convolutional Neural Network			2020
6	CeiT-S (384 finetune res)	88.1%		×	Incorporating Convolution Designs into Visual Transformers			2021
7	Assemble ResNet-50	87.82%		×	Compounding the Performance Improvements of Assembled Techniques in a Convolutional Neural Network			2020
8	NASNet-A Large	87.56%		×	Learning Transferable Architectures for Scalable Image Recognition			2017

NAS for Mobile Devices





$$\mathbb{E}[\text{Latency}] = \alpha \times F(\text{conv\_3x3}) + \beta \times F(\text{conv\_5x5}) + \sigma \times F(\text{identity}) + \ldots\ldots + \zeta \times F(\text{pool\_3x3})$$

$$\mathbb{E}[\text{latency}] = \sum_i \mathbb{E}[\text{latency}_i]$$

$$Loss = Loss_{CE} + \lambda_1 ||w||_2^2 + \lambda_2 \mathbb{E}[\text{latency}]$$

Cai, H., Zhu, L., & Han, S. (2018). Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.

NAS for Mobile Devices: Results

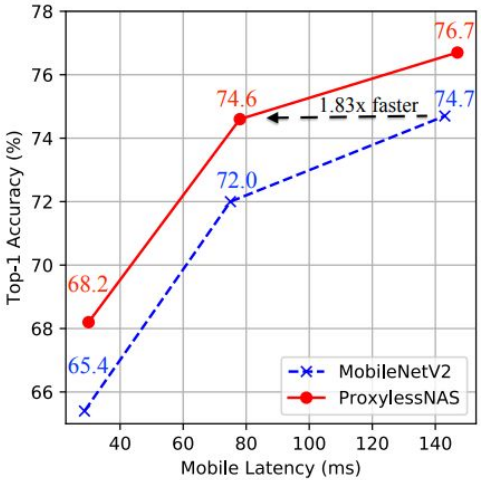


Figure 4: ProxylessNAS consistently outperforms MobileNetV2 under various latency settings.

Cai, H., Zhu, L., & Han, S. (2018). Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.

NAS software

NAS software

1) https://github.com/huawei-noah/vega

vega

AutoML tools chain

Python MIT 105 488 14 1 Updated 2 hours ago

2) https://github.com/microsoft/nni

nni

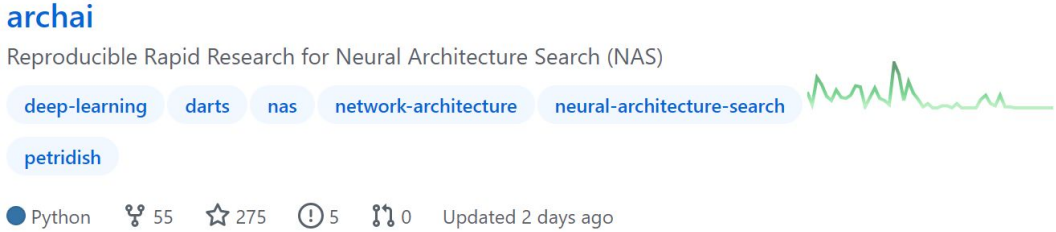
An open source AutoML toolkit for automate machine learning lifecycle, including feature engineering, neural architecture search, model compression and hyper-parameter tuning.

python data-science machine-learning deep-learning neural-network tensorflow machine-learning-algorithms

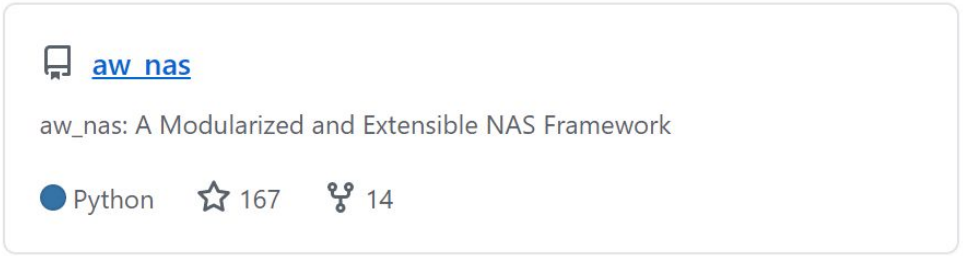
Python MIT 1,311 9,538 249 (16 issues need help) 18 Updated 3 hours ago

NAS software

3) https://github.com/microsoft/archai



4) https://github.com/walkerning/aw_nas



NAS Software

5) https://github.com/On-Point-RND/SeqNAS

Udovichenko I. et al. SeqNAS: Neural architecture search for event sequence classification //IEEE Access. – 2024.

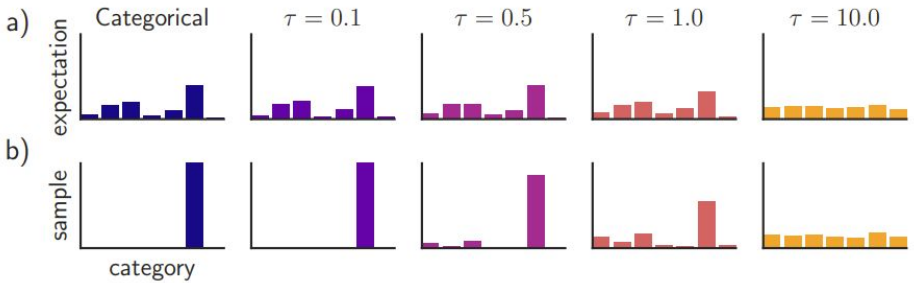
Thank you for attention!

Appendix

Gumbel-softmax reparametrization

- z - is a categorical distribution with probabilities $\pi_1, \pi_2, ...\pi_k$
  $\mathbb{E}_p[z] = [\pi_1, ..., \pi_k]$
- Gumbel-max trick: $z = \text{one\_hot}\left(\arg\max_i [g_i + \log \pi_i]\right)$
  where $g_1 ... g_k$ are i.i.d samples drawn from Gumbel$(0, 1)$[1]
- One-hot-encoding is approximated via softmax:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)}$$



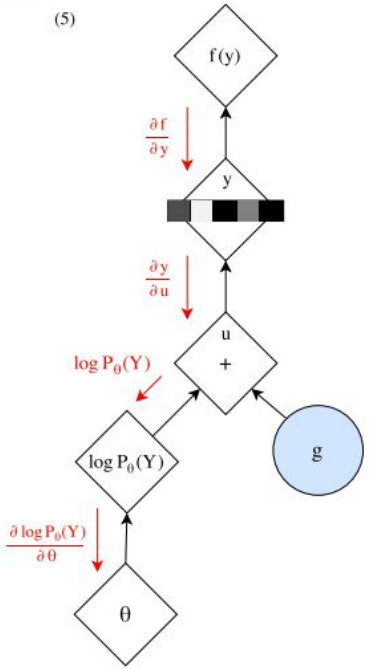
Jang, E., Gu, S., & Poole, B. (2016). Categorical reparameterization with gumbel-softmax.

Gumbel-softmax reparametrization

For distributions that are reparameterizable, we can compute the sample z as a deterministic function g of the parameters θ and an independent random variable ϵ, so that $z = g(\theta, \epsilon)$. The path-wise gradients from f to θ can then be computed without encountering any stochastic nodes:

$$\frac{\partial}{\partial \theta}\mathbb{E}_{z\sim p_\theta}\left[f(z)\right] = \frac{\partial}{\partial \theta}\mathbb{E}_{\epsilon}\left[f(g(\theta,\epsilon))\right] = \mathbb{E}_{\epsilon\sim p_\epsilon}\left[\frac{\partial f}{\partial g}\frac{\partial g}{\partial \theta}\right] \quad (5)$$

Gradient backprop:



Neural Tangent Kernel

- NTK is useful to describe training of a neural network of infinite width limit.
- Fact: for a ***large*** network ***relative*** changes of weights during training is small. $\frac{\|\boldsymbol{w}(n) - \boldsymbol{w}_0\|_2}{\|\boldsymbol{w}_0\|_2}$
- Infinite width neural network is approximately a kernel linear regression:

$$f(x, \boldsymbol{w}) \approx f(x, \boldsymbol{w_0}) + \nabla_w f(x, \boldsymbol{w_0})^T (\boldsymbol{w} - \boldsymbol{w_0})$$

$$\boldsymbol{y}(\boldsymbol{w}) \approx y(\boldsymbol{w_0}) + \nabla_w \boldsymbol{y}(\boldsymbol{w_0})^T (\boldsymbol{w} - \boldsymbol{w_0})$$

With a feature map: $\boldsymbol{\phi}(\boldsymbol{x}) = \nabla_w f(\boldsymbol{x}, \boldsymbol{w_0})$

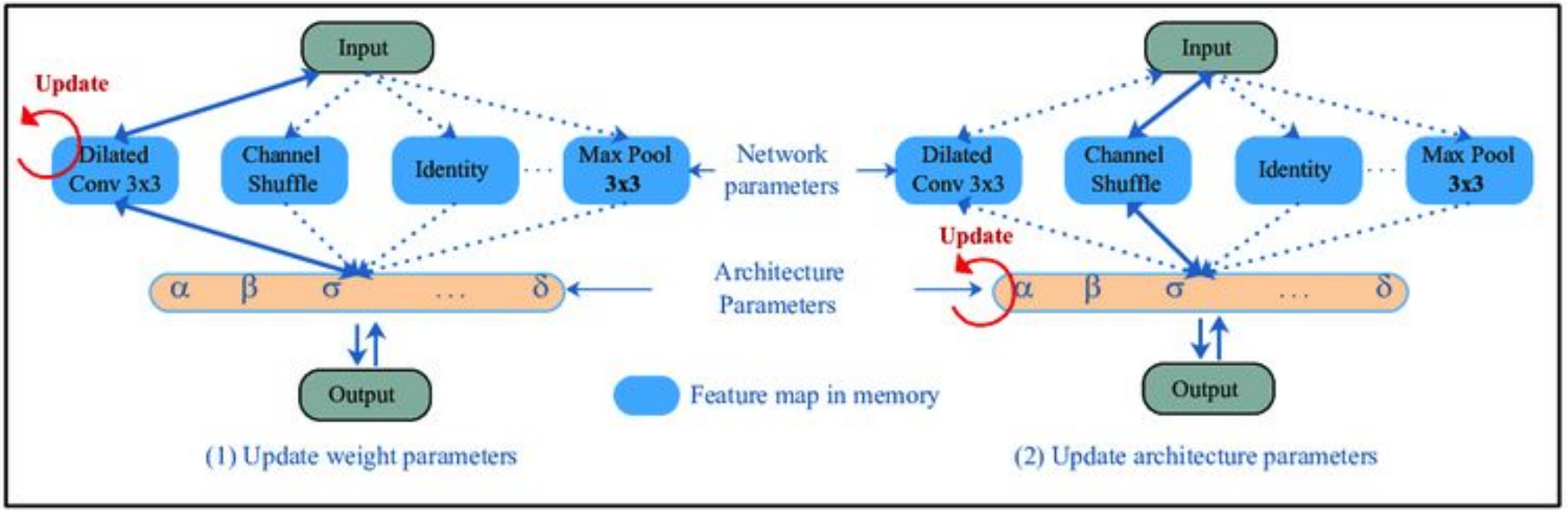
- Neural tangent kernel: $\Theta(x, x') = \nabla_w f(x, w_0) \nabla_w f(x', w_0)^T$
- Training can be approximated in a continuous time:

$$\mathbf{y}(\mathbf{w}(t)) = \mathbf{u}(t) + \overline{\mathbf{y}}$$

$$\mathbf{u}(t) \approx \mathbf{u}(0) \exp^{-\eta \, \Theta_{w_0} t}$$

Jacot A., Gabriel F., Hongler C. Neural tangent kernel: Convergence and generalization in neural networks //Advances in neural information processing systems. – 2018.

see: https://rajatvd.github.io/NTK/



(b) DARTS update strategy

At each iteration, the training the procedure includes 2 alternating optimizers' steps.

- Firstly, the network weights are updated by taking a gradient step in a direction of minimizing loss on a training data, considering multilayers' weights as fixed.
- Secondly, the weights for multilayers (architecture parameters) trained on validation data

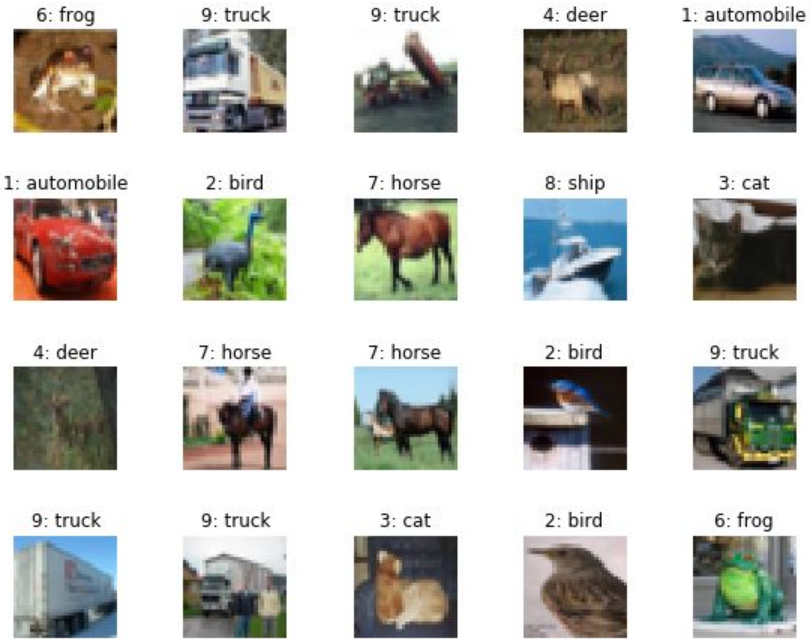
Object classification: CIFAR-10 dataset

Object classification problem

10 classes

50,000 train

10,000 test



DARTS - Accuracy 97.17%

Reproduced results from the paper

Penn Tree Bank dataset & problem

Penn Tree Bank (PTB) is a text corpus with 10000 unique terms.

```
memotec mlx nahb punts rake regatta rubens sim snack-food ssangyong swapo wachter <eos> pierre
<unk> N years old will join the board as a nonexecutive director nov. N <eos> mr. <unk> is
chairman of <unk> n.v. the dutch publishing group <eos> rudolph <unk> N years old and former
chairman of consolidated gold fields plc was named a nonexecutive director of this british
```

Excerpt from the PTB dataset

Language modeling problem: need to predict the next term in the text based on previous ones.

$$e^{-\frac{1}{N}\sum_i \ln p_{w_i}}$$

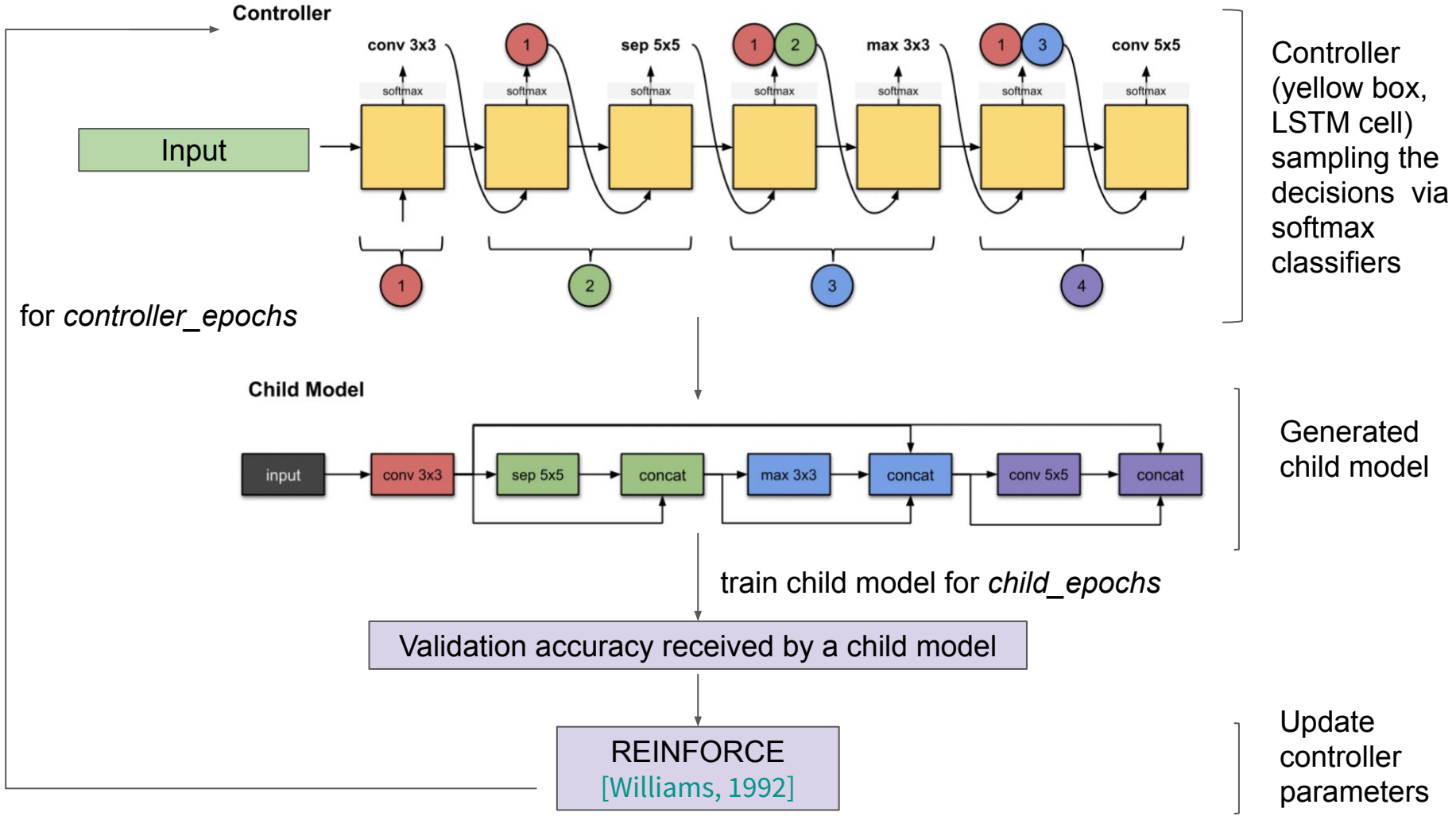
Common quality metric: perplexity

PTB dataset: https://github.com/tomsercu/lstm/tree/master/data

Current state-of-the-art NAS methods

- **ENAS**: (Zoph et al. 2018, In Proceedings of the IEEE conference on computer vision and pattern recognition)
- **DARTS:** Differentiable Architecture Search (Liu et al. 2018; accepted at ICLR'19)
- **ProxylessNAS:** Direct Neural Architecture Search on Target Task and Hardware (Cai et al. 2018; accepted at ICLR'19)
- **MnasNet:** Platform-Aware Neural Architecture Search for Mobile (Tan et al. 2018; accepted at CVPR'19)
- **SNAS:** Stochastic Neural Architecture Search (Xie et al. 2018; accepted at ICLR'19)
- **AmoebaNet-B:** Regularized Evolution for Image Classifier Architecture Search (Real et al. 2018, Accepted for publication at AAAI 2019)

Let us concentrate on DARTS, ENAS, Regularized evolution as the main NAS state-of-the-art methods.


Controller
conv 3x3
sep 5x5
max 3x3
conv 5x5
softmax
Input
1
2
3
4
for *controller_epochs*
Controller (yellow box, LSTM cell) sampling the decisions via softmax classifiers
Child Model
input
conv 3x3
sep 5x5
concat
max 3x3
concat
conv 5x5
concat
Generated child model
train child model for *child_epochs*
Validation accuracy received by a child model
REINFORCE
[Williams, 1992]
Update controller parameters