

# Направление: Нелинейная цифровая обработка сигналов.

Тема: Поиск нейросетевых архитектур.  
Neural architecture search (NAS).

Авторы: к.т.н., старший инженер Huawei Антонов Лев  
старший инженер Huawei Власов Роман.

# История AutoML

TOM SIMONITE BUSINESS 18.13.2017 07:00 AM

## Google's Learning Software Learns to Write Learning Software

Google's researchers have taught machine-learning software to build machine-learning software, in a project dubbed AutoML.

MIT  
Technology  
Review

Artificial intelligence / Machine learning

## Google's self-training AI turns coders into machine-learning masters

Automating the training of machine-learning systems could make AI much more accessible.

by Will Knight

January 17, 2018

TECH

## Google Researchers Are Teaching Their AI to Build Its Own, More Powerful AI

DAVID NIELD 19 MAY 2017

Google has announced another big push into artificial intelligence, unveiling a new approach to machine learning where [neural networks](#) are used to build better neural networks - essentially teaching AI to teach itself.

В конце 2017 / начале 2018 в ведущих технологических изданиях возник внезапный ажиотаж вокруг создания компанией Google некоего ПО, генерирующего нейронные сети или даже другое ПО.

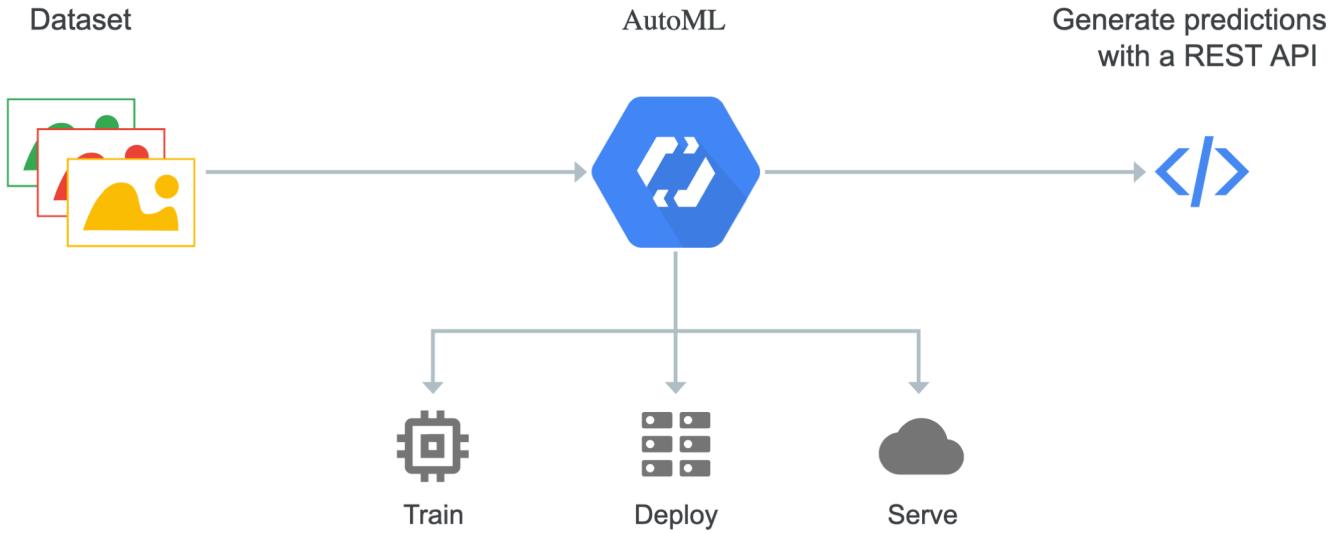
## Google's AutoML lets you train custom machine learning models without having to code

Frederic Lardinois @fredericl / 5:00 pm MSK • January 17, 2018

# Google AutoML

На самом деле Google представила облако автоматизации машинного обучения, вычислительным ядром которого является AutoML.

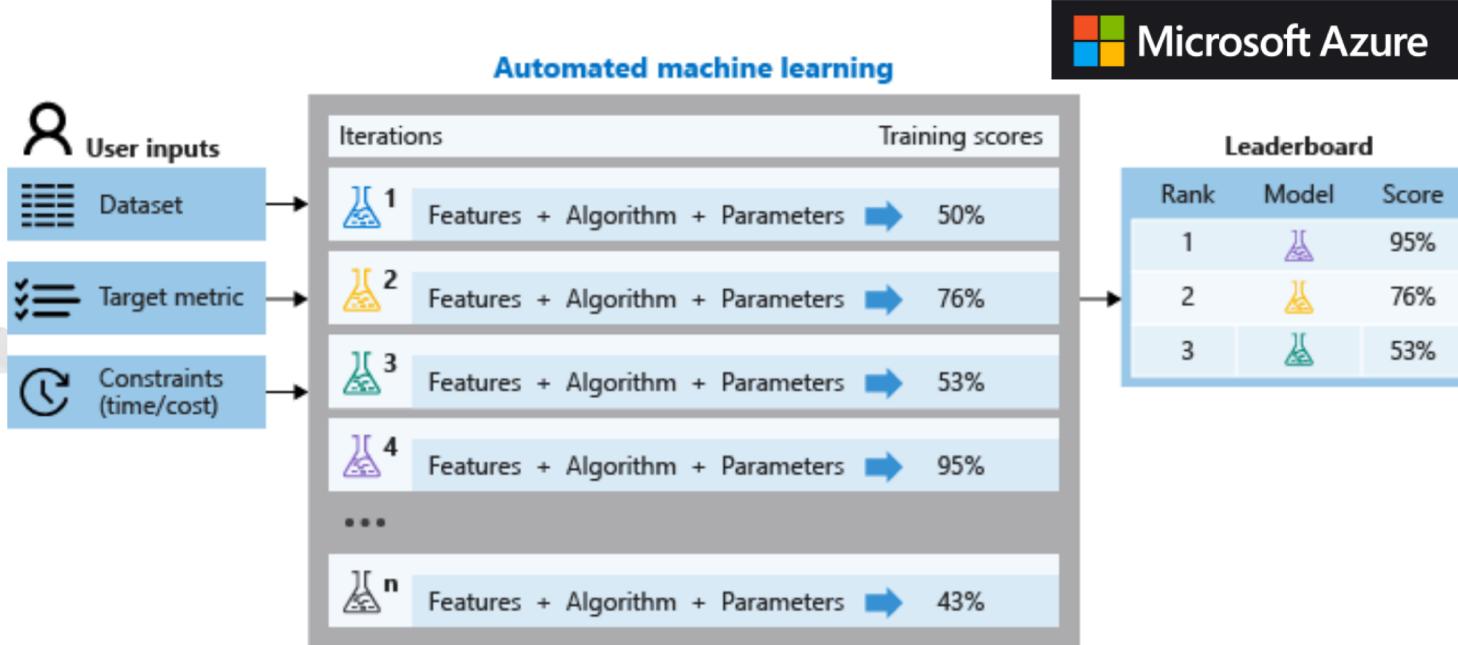
## How AutoML works



<https://cloud.google.com/automl>

# Microsoft Azure AutoML

И не только Google...



<https://docs.microsoft.com/ru-ru/azure/machine-learning/concept-automated-ml>

# Представление AutoML

## AutoML

Говоря об автоматизации машинного обучения, стоит заметить, что это направление включает в себя следующие задачи:

- Сбор данных.
- Исследование данных.
- Отбор и выделение признаков.
- Создание и отбор модели(ей).
- Обучение модели.
- Развёртывание готового решения.

# Представление AutoML

AutoML

NAS  
Автоматизация  
построения модели

Говоря об автоматизации машинного обучения, стоит заметить, что это направление включает в себя следующие задачи:

- Сбор данных.
- Исследование данных.
- Отбор и выделение признаков.
- **Создание и отбор моделей.**
- Обучение модели.
- Развёртывание готового решения.

С самого начала AutoML и Neural Architecture Search (NAS) стали синонимами, хотя по факту, это не одно и тоже.

# Представление AutoML

**AutoML**

**NAS**  
Автоматизация  
построения модели

Цель NAS: Автоматизация построения архитектур.



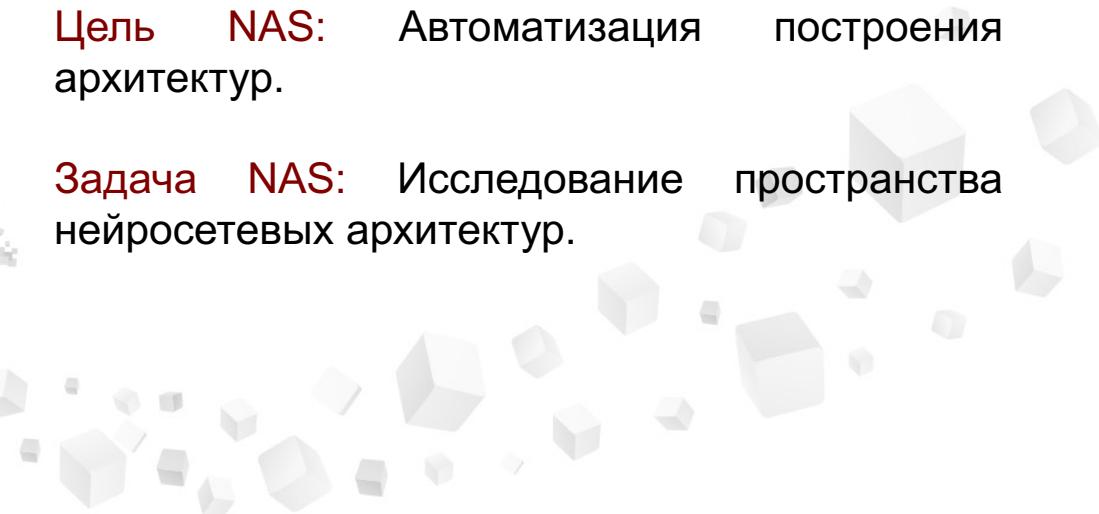
# Представление AutoML

**AutoML**

**NAS**  
Автоматизация  
построения модели

**Цель NAS:** Автоматизация построения архитектур.

**Задача NAS:** Исследование пространства нейросетевых архитектур.



# Представление AutoML

## AutoML

Оптимизации  
гиперпараметров

**NAS**  
Автоматизация  
построения модели

Цель NAS: Автоматизация построения архитектур.

Задача NAS: Исследование пространства нейросетевых архитектур.

**NAS** – это частный случай оптимизации гиперпараметров.

# Представление AutoML

AutoML

Оптимизации  
гиперпараметров

NAS  
Автоматизация  
построения модели

Цель NAS: Автоматизация построения архитектур.

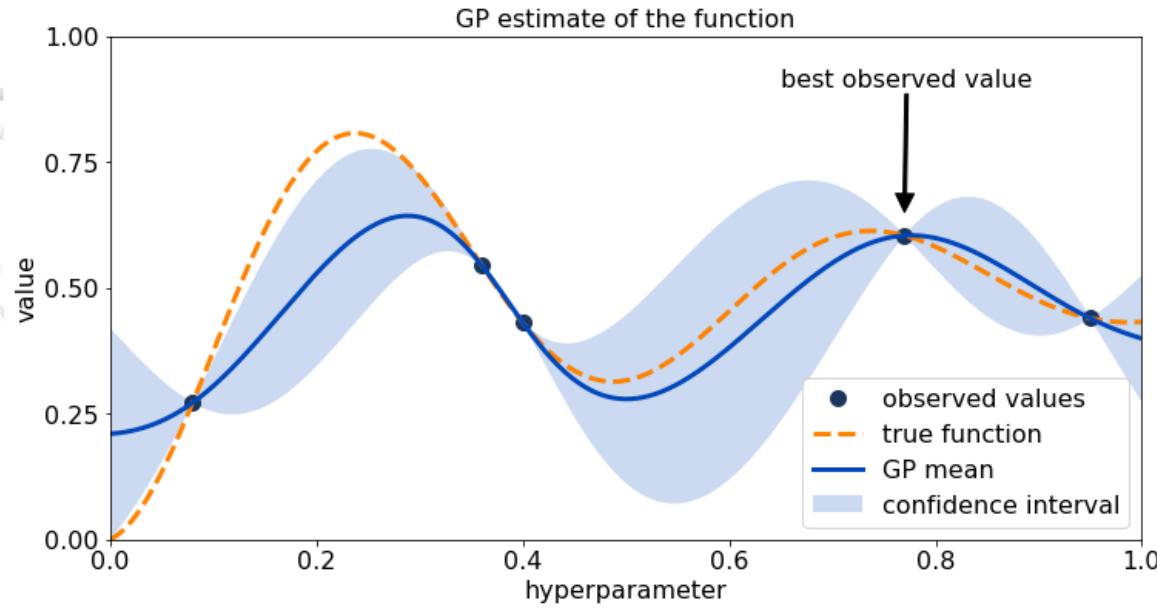
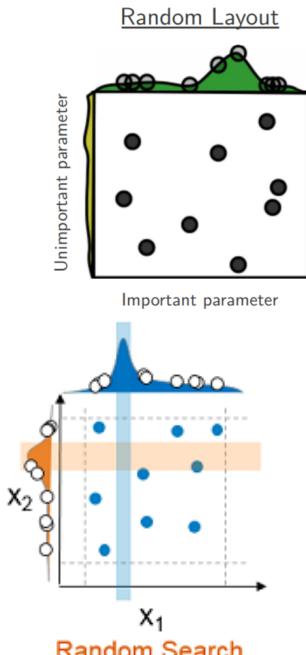
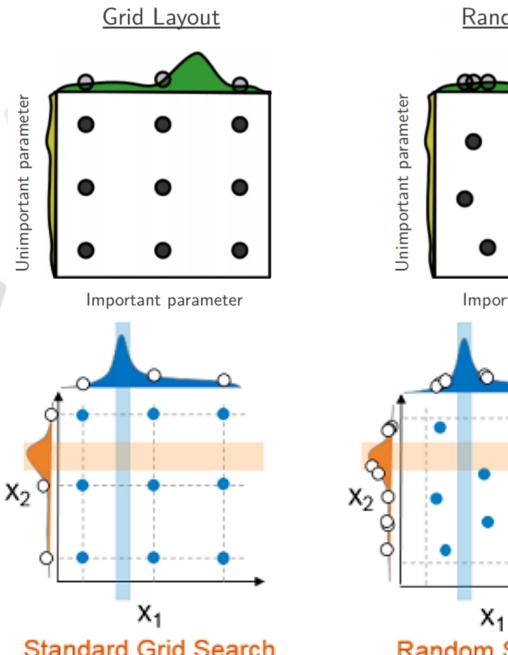
Задача NAS: Исследование пространства нейросетевых архитектур.

NAS – это частный случай оптимизации гиперпараметров.

Было необходимо понять, какие подходы дали хороший результат для **HyperparamOpt**, чтобы попытаться применить их для более узкой области NAS.

# Оптимизация гиперпараметров

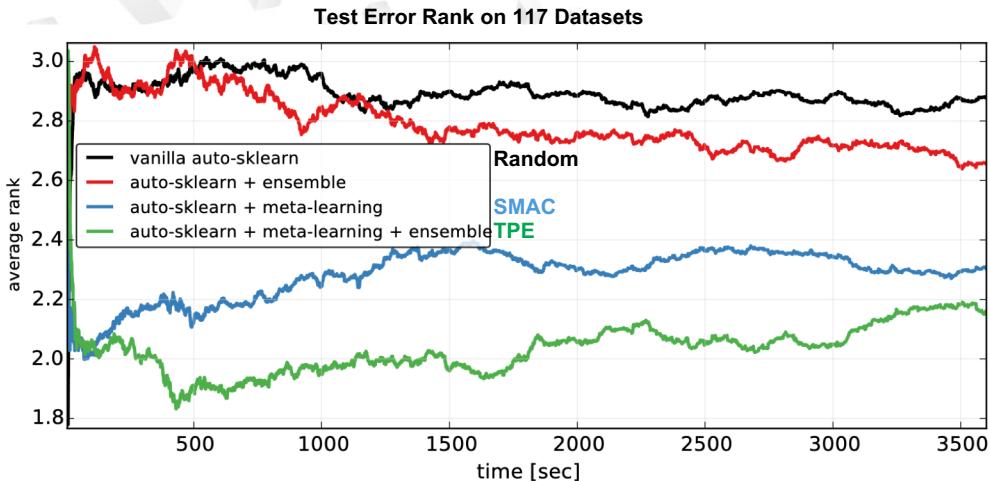
Начало прогресса в этой области пришлось на 2012-2015 годы, когда широко начала применяться Байесовская оптимизация.



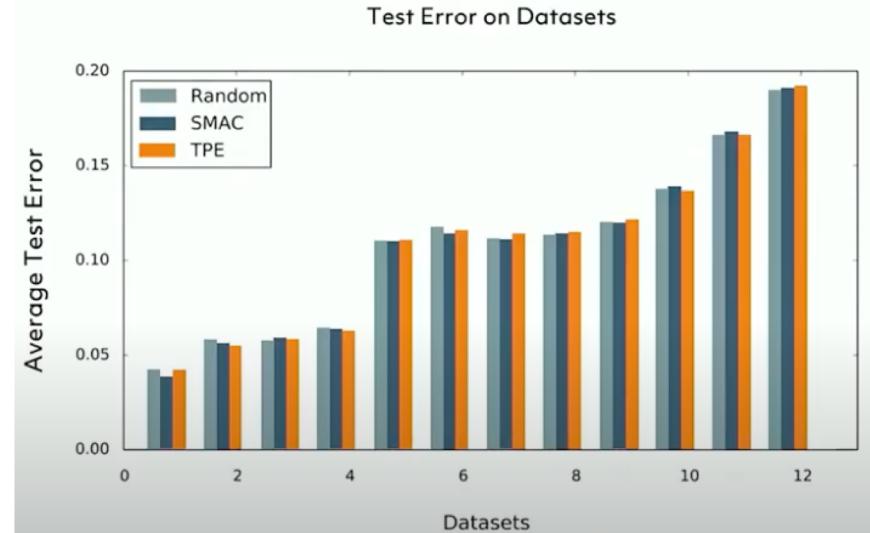
<https://www.sicara.ai/blog/2019-14-07-determine-network-hyper-parameters-with-bayesian-optimization>

# Оптимизация гиперпараметров

Основные результаты сравнения алгоритмов оптимизации гиперпараметров в статье "Efficient and Robust Automated Machine Learning" с конференции NIPS-2015.



Основные результаты сравнения алгоритмов оптимизации гиперпараметров из презентации Determined AI 2019.



<https://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning>

<https://determined.ai/recording/random-search-and-reproducibility/>

# Оптимизация гиперпараметров

## Grid search [edit]

- Determined<sup>[2]</sup>, a DL Training Platform includes grid search for PyTorch and TensorFlow (Keras and Estimator) models.
- H2O AutoML<sup>[3]</sup> provides grid search over algorithms in the H2O open source machine learning library.
- Katib<sup>[4]</sup> is a Kubernetes-native system that includes grid search.
- scikit-learn is a Python package that includes grid<sup>[5]</sup> search.
- Talos<sup>[6]</sup> includes grid search for Keras.
- Tune<sup>[7]</sup> is a Python library for distributed hyperparameter tuning and supports grid search.

## Random search [edit]

- Determined<sup>[2]</sup> is a DL Training Platform that supports random search for PyTorch and TensorFlow (Keras and Estimator) models.
- hyperopt<sup>[8]</sup>, also via hyperas<sup>[9]</sup> and hyperopt-sklearn<sup>[10]</sup>, are Python packages which include random search.
- Katib<sup>[4]</sup> is a Kubernetes-native system that includes random search.
- scikit-learn is a Python package which includes random<sup>[11]</sup> search.
- Talos<sup>[6]</sup> includes a customizable random search for Keras.
- Tune<sup>[7]</sup> is a Python library for distributed hyperparameter tuning and supports random search over arbitrary parameter distributions.

## Bayesian [edit]

- Auto-sklearn<sup>[25]</sup> is a Bayesian hyperparameter optimization layer on top of scikit-learn.
- Ax<sup>[26]</sup> is a Python-based experimentation platform that supports Bayesian optimization and bandit optimization as exploration strategies.
- BOCS<sup>[27]</sup> is a Matlab package which uses semidefinite programming for minimizing a black-box function over discrete inputs.<sup>[27]</sup> A Python 3 implementation is also included.
- HpBandSter<sup>[28]</sup> is a Python package which combines Bayesian optimization with bandit-based methods.<sup>[28]</sup>
- Katib<sup>[4]</sup> is a Kubernetes-native system which includes bayesian optimization.
- mirlMBO<sup>[29]</sup>, also with mirl<sup>[30]</sup>, is an R package for model-based/Bayesian optimization of black-box functions.
- optuna<sup>[31]</sup> is a Python package for black box optimization, compatible with arbitrary functions that need to be optimized.
- scikit-optimize<sup>[32]</sup> is a Python package or sequential model-based optimization with a scipy.optimize interface.<sup>[29]</sup>
- SMAC<sup>[33]</sup> SMAC is a Python/Java library implementing Bayesian optimization.<sup>[30]</sup>
- tuneRanger<sup>[34]</sup> is an R package for tuning random forests using model-based optimization.

## Gradient-based optimization [edit]

- FAR-HO<sup>[35]</sup> is a Python package containing Tensorflow implementations and wrappers for gradient-based hyperparameter optimization with forward and backward passes.
- XGBoost<sup>[36]</sup> is an open-source software library that provides a gradient boosting framework for C++, Java, Python, R, and Julia.

## Evolutionary [edit]

- deap<sup>[37]</sup> is a Python framework for general evolutionary computation which is flexible and integrates with parallelization packages like scoop<sup>[38]</sup> and pypy<sup>[39]</sup>.
- Determined<sup>[2]</sup> is a DL Training Platform that supports PBT for optimizing PyTorch and TensorFlow (Keras and Estimator) models.
- devol<sup>[40]</sup> is a Python package that performs Deep Neural Network architecture search using genetic programming.
- nevergrad<sup>[41]</sup> is a Python package which includes Differential\_evolution, Evolution\_strategy, Bayesian\_optimization, population control methods for the noisy case and Particle\_swarm\_optimization.<sup>[32]</sup>
- Tune<sup>[7]</sup> is a Python library for distributed hyperparameter tuning and leverages nevergrad<sup>[42]</sup> for evolutionary algorithm support.

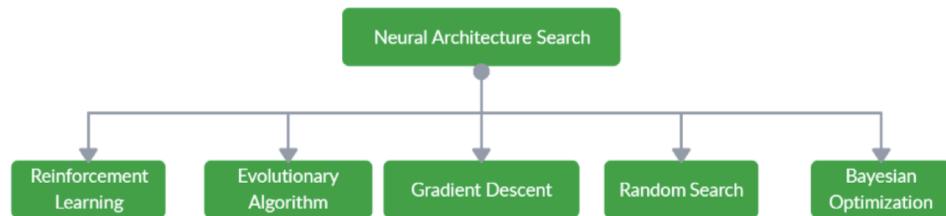
## Early Stopping [edit]

[https://en.wikipedia.org/wiki/Hyperparameter\\_optimization](https://en.wikipedia.org/wiki/Hyperparameter_optimization)

<https://blog.ml6.eu/automated-machine-learning-using-auto-keras-f9b7002aee0e>

▪ Перечень основных фреймворков оптимизации гиперпараметров и лежащих в их основе алгоритмах по версии Википедии.

▪ Очевидно, что именно эти подходы станут основными при исследовании пространства архитектур в NAS.



# Развитие идеи NAS

- Первая попытка автоматизации архитектурного поиска была предпринята в работе *Lingxi Xie, Alan Yuille “Genetic CNN” (Dept. of Comput. Sci., Johns Hopkins Univ., Baltimore, MD, USA) 2015.*

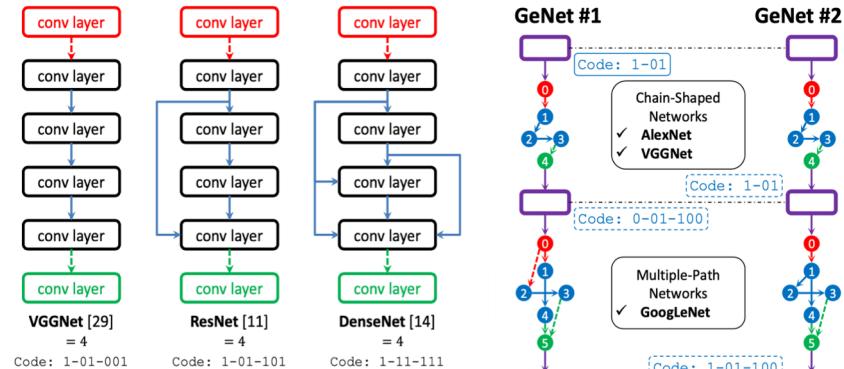
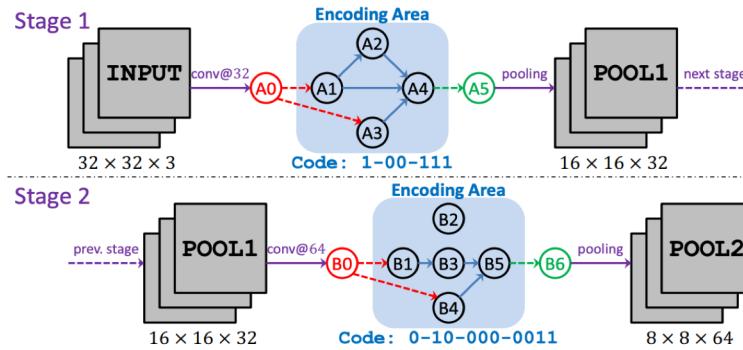


Figure 2. The basic building blocks of VGGNet [32], ResNet [13] and a variant of DenseNet [15] can be encoded as binary strings defined in Section 3.1.

- Архитектура сети задается с помощью бинарного массива.
- С помощью подобного представления закодировали все самые известные на тот момент архитектуры **VGG**, **ResNet**, **DenceNet**.
- Параметры элементов модели фиксированы.
- Результаты не самые выдающиеся: найдена архитектура, с результатом как у VGG для датасета CIFAR10.

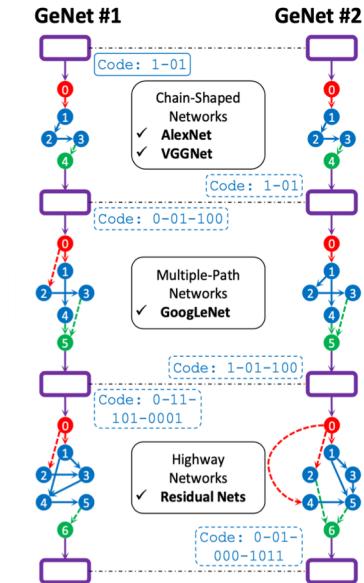
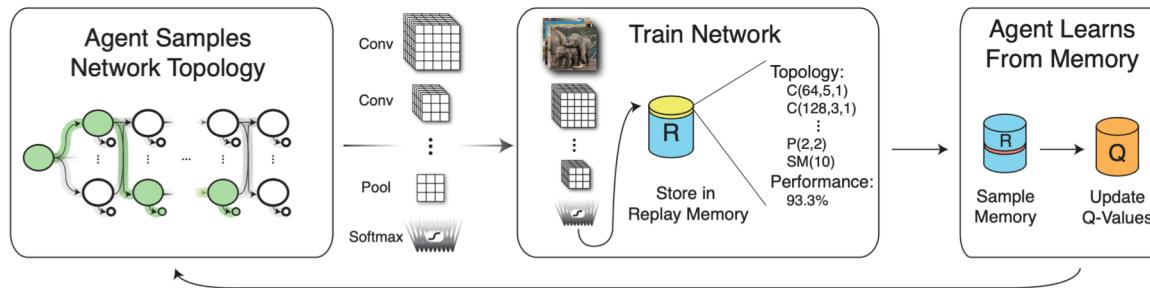


Figure 5. Two network structures learned from the two independent genetic processes on the CIFAR10 dataset (best viewed in color). These are three-stage networks ( $S = 3$ ) with  $(K_1, K_2, K_3) = (3, 4, 5)$ .

# Развитие идеи NAS

- Первая попытка применения обучения с подкреплением в NAS в работе *Bowen Baker, Otkrist Gupta, Nikhil Naik, Ramesh Raskar “Designing Neural Network Architectures using Reinforcement Learning” (Media Laboratory, Massachusetts Institute of Technology, Cambridge MA 02139, USA) 2016.*



$$Q_{t+1}(s_i, u) = (1 - \alpha)Q_t(s_i, u) + \alpha [r_t + \gamma \cdot \max_{u' \in U(s_j)} Q_t(s_j, u')]$$

- Сгенерированные нейронные сети обучаются (среда) и возвращают некоторый **score**, который служит наградой Агенту, за полезную работу.
- Агент обучается и совершает действия используя **Epsilon-Greedy Algorithm**, в основе которого лежит **Q-function (уравнение Бэлмана)**.

- Существует некоторая **среда**, в которой **зафиксированы состояния** и **действия** для перехода в новое состояние.
- Существует **Агент**, который, совершая набор действий, **создает нейронную архитектуру**.

<https://ieeexplore.ieee.org/document/8237416>

HUAWEI TECHNOLOGIES CO., LTD.

# Развитие идеи NAS

Bowen Baker, Otkrist Gupta, Nikhil Naik, Ramesh Raskar “*Designing Neural Network Architectures using Reinforcement Learning*” (Media Laboratory, Massachusetts Institute of Technology, Cambridge MA 02139, USA) 2016.

Layer Type	Layer Parameters	Parameter Values
Convolution (C)	$i \sim$ Layer depth $f \sim$ Receptive field size $\ell \sim$ Stride $d \sim$ # receptive fields $n \sim$ Representation size	$< 12$ Square. $\in \{1, 3, 5\}$ Square. Always equal to 1 $\in \{64, 128, 256, 512\}$ $\in \{(\infty, 8], (8, 4], (4, 1]\}$
Pooling (P)	$i \sim$ Layer depth $(f, \ell) \sim$ (Receptive field size, Strides) $n \sim$ Representation size	$< 12$ Square. $\in \{(5, 3), (3, 2), (2, 2)\}$ $\in \{(\infty, 8], (8, 4] \text{ and } (4, 1]\}$
Fully Connected (FC)	$i \sim$ Layer depth $n \sim$ # consecutive FC layers $d \sim$ # neurons	$< 12$ $< 3$ $\in \{512, 256, 128\}$
Termination State	$s \sim$ Previous State $t \sim$ Type	Global Avg. Pooling/Softmax

Table 1: **Experimental State Space.** For each layer type, we list the relevant parameters and the values each parameter is allowed to take.

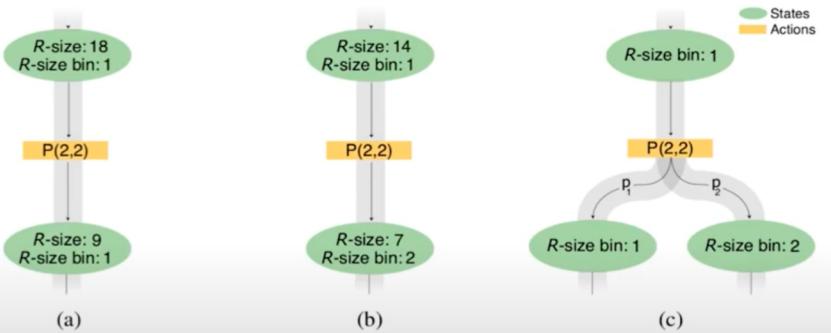
- Отдельно в статье рассматриваются сами **состояния**. Это обусловлено тем, что перебрать все возможные варианты состояний фактически невозможно. Нужно ограничиться каким-то набором элементов и параметров.

# Развитие идеи NAS

<https://ieeexplore.ieee.org/document/8237416>

Bowen Baker, Otkrist Gupta, Nikhil Naik, Ramesh Raskar “**Designing Neural Network Architectures using Reinforcement Learning**” (Media Laboratory, Massachusetts Institute of Technology, Cambridge MA 02139, USA) 2016.

## The State Space



- Агент может заканчивать построение сети с любого состояния.
- Из последнего слоя, если достигли максимальной длины, переходим в терминальное состояние.
- Отсутствуют skip-connections.
- Максимальное количество fully-connections не превышает 2.
- Из fully-connection мы можем перейти только в fully-connection или другие состояния, где количество нейронов обязательно должно быть меньше.
- Из convolution можно перейти в любое другое состояние.
- Из pooling можно перейти в любое другое состояние кроме pooling. (2 pooling подряд можно сконкатенировать больший пулинг).
- В fully-connected мы можем переходить, если все 3 размерности текущего слоя (тензора) принадлежат интервалам  $(8, 4]$ ,  $(4, 1]$ .

- Состояния задаются кортежем.
- У каждого кортежа есть уровень слоя ( $i$  – Layer depth). В данном подходе запрещены все skip-connections и из уровня (слоя)  $i - 1$  мы можем идти только в уровень  $i$ .

## The Action Space

- Agent can terminate a path at any point
- Transitions for a state with layer depth  $i$  to a state with layer depth  $i + 1$
- Any state at the maximum layer depth, may only transition to a termination layer
- Limit the number of fully connected layers 2
- The agent at a state with type FC may transition to another state with type FC if and only if the number of consecutive FC states is less than the maximum allowed
- A state  $s$  of type FC with number of neurons  $d$  may only transition to either a termination state or a state  $s'$  of type FC with number of neurons  $d' \leq d$
- An agent at a state of type convolution may transition to a state with any other layer type
- An agent at a state with layer type pooling (P) may transition to a state with any other layer type other than another P
- Only states with representation size in bins  $(8, 4]$  and  $(4, 1]$  may transition to an FC layer

# Развитие идеи NAS

- Отправная точка развития идеи NAS в работе *Barret Zoph, Quoc V. Le “Neural Architecture Search with Reinforcement Learning” (Google) 2016.*

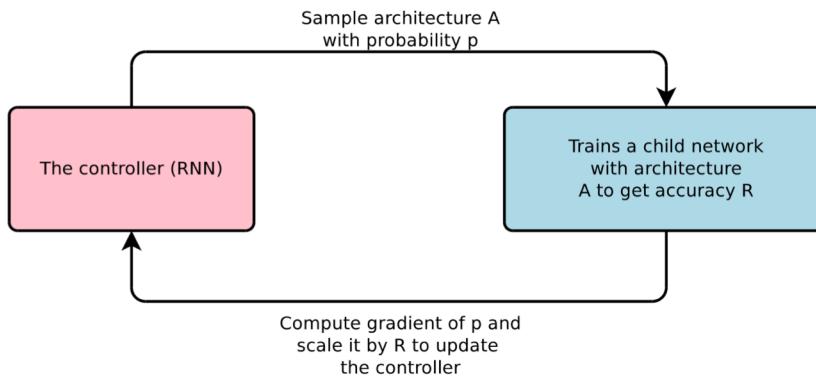


Figure 1: An overview of Neural Architecture Search.

- В данной статье предлагается заменить обучение с подкреплением на **глубокое обучение с подкреплением**.
- В качестве **агента (контроллера)** выступает сеть **RNN (LSTM)**.
- Агент генерирует сеть **A** (с какой-то вероятностью).
- Нейросеть **A** обучается (**среда взаимодействия**) и возвращает **score R** (**reward / награда**).

- **Обновление параметров контроллера** происходит на основе **возвращенной награды R из среды взаимодействия**.

# Развитие идеи NAS

<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/neural-architecture-search-limitations-and-extensions-8141bec7681f>

Barret Zoph, Quoc V. Le “**Neural Architecture Search with Reinforcement Learning**” (Google) 2016.

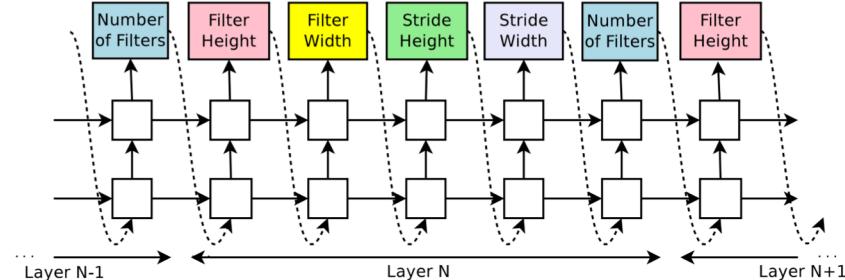


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step as input.

- На иллюстрации рассматривается пример генерации только **convolution** слоев.
- Для слоя **N RNN (LSTM)** генерирует 5 значений:
  - - высоту фильтра;
  - - длину фильтра;
  - - высоту шага;
  - - длину шага;
  - - количество фильтров в слое.
- И так для каждого слоя.

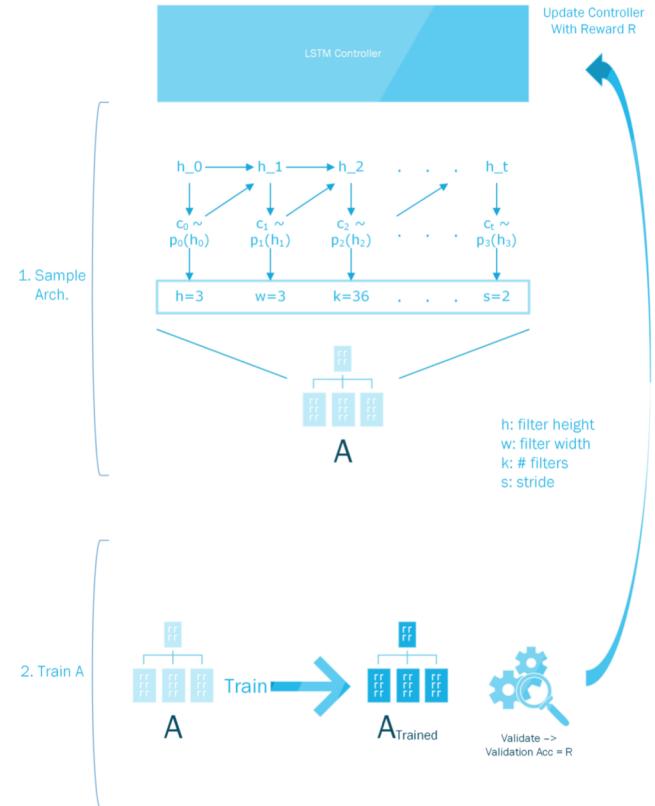
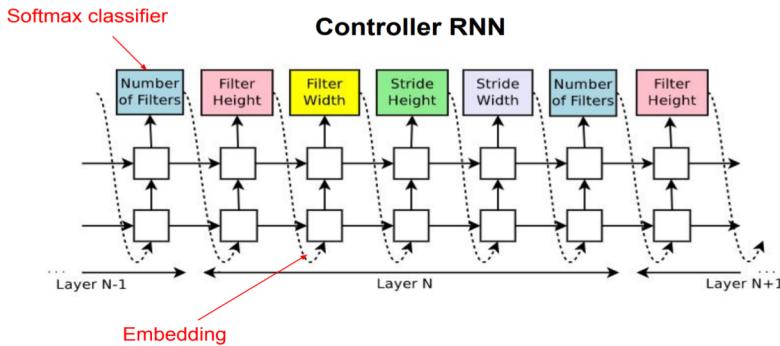


Схема работы NAS

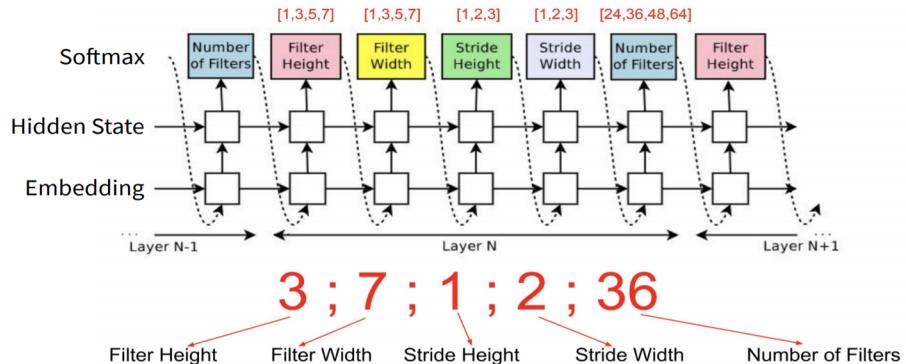
# Развитие идеи NAS

Barret Zoph, Quoc V. Le “**Neural Architecture Search with Reinforcement Learning**” (Google) 2016.

Neural Architecture Search for Convolutional Networks



Training with REINFORCE (Zoph and Le, 2017)



- На иллюстрации рассматривается пример генерации только **convolution** слоев.

- Для слоя **N** наша **RNN (LSTM)** генерирует нам 5 значений:

- высоту фильтра;
- длину фильтра;
- высоту шага;
- длину шага;
- количество фильтров в слое.

- И так для каждого слоя.

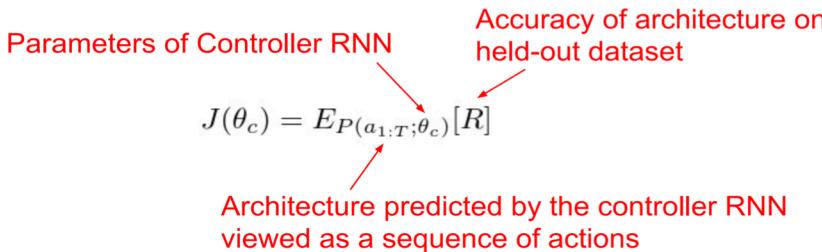
- На выходе каждого слоя стоит **SoftMax**, который возвращает нам вектор “вероятностей” какого-то определенного параметра элементов данного слоя.

- Для каждого слоя **Контроллер** может выбрать высоту фильтра из [1, 3, 5, 7], ширину фильтра [1, 3, 5, 7] и количество фильтров в слое из [24, 36, 48, 64] (похоже на поиск в вероятностной сетке).

# Развитие идеи NAS

Barret Zoph, Quoc V. Le “**Neural Architecture Search with Reinforcement Learning**” (Google) 2016.

## Как корректировать политику RNN (обучение контроллера)?



- У рекуррентной нейронной сети есть параметры  $\theta_c$ . И задача контроллера – максимизировать **Reward**, полученный сгенерированной нейронной сетью.



<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/neural-architecture-search-limitations-and-extensions-8141bec7681f>

[https://metalearning-cvpr2019.github.io/assets/CVPR\\_2019\\_Metalearning\\_Tutorial\\_Nikhil\\_Naik.pdf](https://metalearning-cvpr2019.github.io/assets/CVPR_2019_Metalearning_Tutorial_Nikhil_Naik.pdf)

# Развитие идеи NAS

Barret Zoph, Quoc V. Le “**Neural Architecture Search with Reinforcement Learning**” (Google) 2016.

## Как корректировать политику RNN (обучение контроллера)?

### Training with REINFORCE

Parameters of Controller RNN      Accuracy of architecture on held-out dataset

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

Architecture predicted by the controller RNN viewed as a sequence of actions

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

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

- У рекуррентной нейронной сети есть параметр  $\theta_c$ . И задача контроллера – максимизировать **Reward**, полученный сгенерированной нейронной сетью.
- Здесь  $T$  – количество действий, которое совершил контроллер (выбор высоты, ширины – это все отдельные действия).
- $a_i$  – конкретное действие, которое сделал контроллер.
- $P(a_{1\dots T}, \theta_c)$  – вероятность набора действий  $a$  при заданном наборе параметров **RNN**  $\theta_c$ .
- Обновляем веса контроллера с помощью обычного алгоритма **reinforce**, заменяя сумму мат. ожидания двойной суммой, где  $t$  – это количество сгенерированных нейронных сетей.

<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/neural-architecture-search-limitations-and-extensions-8141bec7681f>

[https://metalearning-cvpr2019.github.io/assets/CVPR\\_2019\\_Metalearning\\_Tutorial\\_Nikhil\\_Naik.pdf](https://metalearning-cvpr2019.github.io/assets/CVPR_2019_Metalearning_Tutorial_Nikhil_Naik.pdf)

# Развитие идеи NAS

Barret Zoph, Quoc V. Le “*Neural Architecture Search with Reinforcement Learning*” (Google) 2016.

**Недостатком алгоритма является то, что огромное время уходит на обучение всех сгенерированных моделей.**

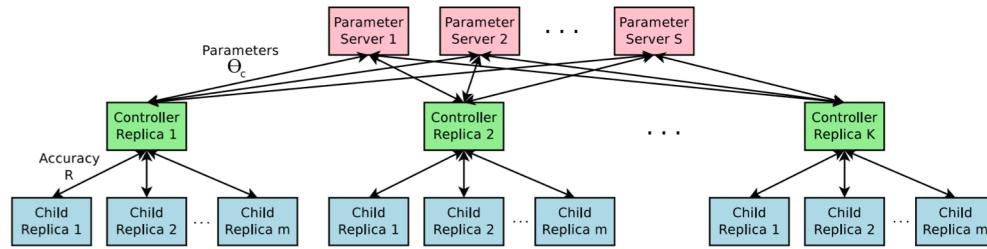


Figure 3: Distributed training for Neural Architecture Search. We use a set of  $S$  parameter servers to store and send parameters to  $K$  controller replicas. Each controller replica then samples  $m$  architectures and run the multiple child models in parallel. The accuracy of each child model is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

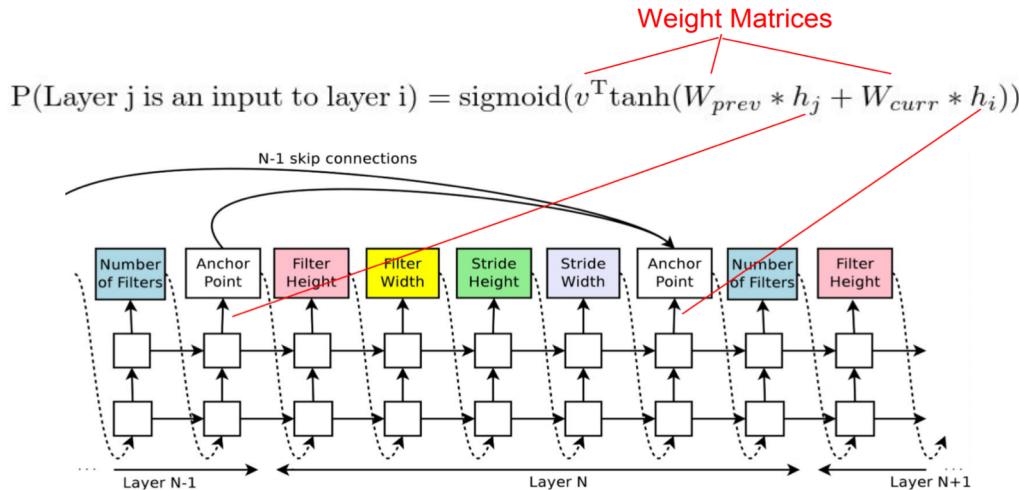
Поэтому в работе предлагается способ распараллеливания алгоритма, основанный на установке копий контроллера на разные сервера и периодическом обмене параметрами между ними.

# Развитие идеи NAS

Barret Zoph, Quoc V. Le “**Neural Architecture Search with Reinforcement Learning**” (Google) 2016.

Еще одним важным моментом является введение в архитектуру skip-connections.

## Neural Architecture Search for CIFAR-10



- В **RNN** добавляется еще одно скрытое состояние **Anchor point** (элемент-действие) для каждого слоя генерируемой архитектуры.
- Далее специальный механизм **attention** принимает решение, какие слои для текущего будут являться входными.

Google

<https://arxiv.org/abs/1611.01578>

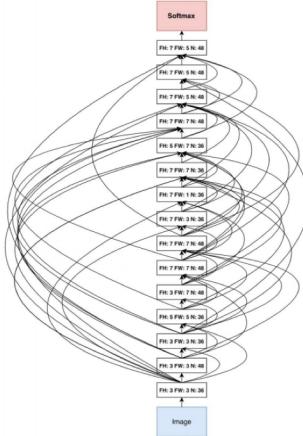
[http://rll.berkeley.edu/deeprlcoursesp17/docs/quoc\\_barret.pdf](http://rll.berkeley.edu/deeprlcoursesp17/docs/quoc_barret.pdf)

# Развитие идеи NAS

Barret Zoph, Quoc V. Le “**Neural Architecture Search with Reinforcement Learning**” (Google) 2016.

## Результаты

### Neural Architecture Search for CIFAR-10



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) with Dropout/Drop-path	21 21	38.6M 38.6M	5.22 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 1202	1.7M 10.2M	5.23 4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16 28	11.0M 36.5M	4.81 4.17
ResNet (pre-activation) (He et al., 2016b)	164 1001	1.7M 10.2M	5.46 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

5% faster

- Результаты сравнения **самых удачных решений в 2016 году** для набора данных **CIFAR10**.
- Пространство поиска включало в себя ReLU, Batch-norm и skip-connection между слоями.
- Контроллер RNN содержал 2 LSTM слоя с 35-ю скрытыми состояниями.

Best result of evolution (Real et al, 2017): 5.4%  
Best result of Q-learning (Baker et al, 2017): 6.92%

Google  
<https://arxiv.org/abs/1611.01578>

[http://rll.berkeley.edu/deeprlcoursesp17/docs/quoc\\_barret.pdf](http://rll.berkeley.edu/deeprlcoursesp17/docs/quoc_barret.pdf)

HUAWEI TECHNOLOGIES CO., LTD.

# Развитие идеи NAS

Barret Zoph, Quoc V. Le “**Neural Architecture Search with Reinforcement Learning**” (Google) 2016.

## Результаты

### CIFAR-10 Experiment Details

- Use 100 Controller Replicas each training 8 child networks concurrently
- Method uses 800 GPUs concurrently at one time
- Reward given to the Controller is the maximum validation accuracy of the last 5 epochs squared
- Split the 50,000 Training examples to use 45,000 for training and 5,000 for validation
- Each child model was trained for 50 epochs
- Run for a total of **12,800** child models
- Used curriculum training for the Controller by gradually increasing the number of layers sampled

Google

- Проблема, представленной в “**Neural Architecture Search with Reinforcement Learning**”, заключается в том, что для достижения результата на датасете CIFAR10 алгоритму понадобилось ~32000 часов расчетов на графическом процессоре.
- Именно этот факт стал причиной разработки более эффективного подхода представленного в статье “**Efficient Neural Architecture Search via Parameter Sharing**”.

<https://arxiv.org/abs/1611.01578>

[http://rll.berkeley.edu/deeprlcoursesp17/docs/quoc\\_barret.pdf](http://rll.berkeley.edu/deeprlcoursesp17/docs/quoc_barret.pdf)

HUAWEI TECHNOLOGIES CO., LTD.

# Развитие идеи NAS

- Второй мощный виток развития идеи **RNN + RL для NAS** представлен в статье *Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “Efficient Neural Architecture Search (ENAS) via Parameter Sharing” (Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.*

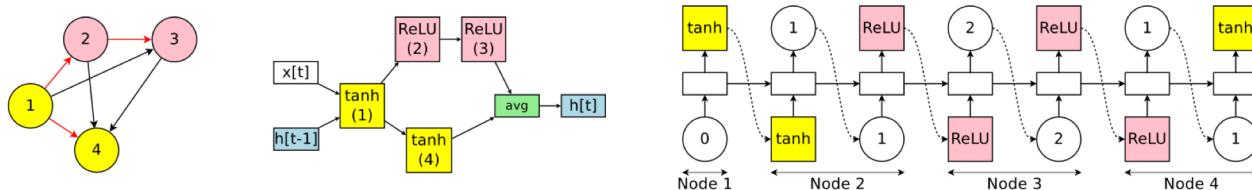


Figure 1. An example of a recurrent cell in our search space with 4 computational nodes. *Left:* The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. *Middle:* The recurrent cell. *Right:* The outputs of the controller RNN that result in the cell in the middle and the DAG on the left. Note that nodes 3 and 4 are never sampled by the RNN, so their results are averaged and are treated as the cell’s output.

Авторы предлагают несколько стратегий архитектурного поиска: **макро- и микропоиск**, совмещая это с техникой **Parameter Sharing**.

- В ENAS задействованы 2 типа нейронных сетей: **Контроллер – RNN (LSTM); Дочерняя модель - синтезируемая CNN для классификации изображений.**
- Как и большинство других алгоритмов NAS, **ENAS** включает в себя 3 основных элемента:
  - Пространство поиска - все возможные архитектуры (дочерние модели), которые могут быть сгенерированы.
  - Стратегия поиска - метод генерации этих архитектур (дочерних моделей).
  - Оценка эффективности - метод измерения эффективности созданных дочерних моделей.

<https://arxiv.org/abs/1802.03268>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “***Efficient Neural Architecture Search (ENAS) via Parameter Sharing***”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

- **Макро поиск** - это подход, при котором контроллер проектирует всю сеть сразу. Примеры публикаций, которые используют этот подход: *NAS on Zoph u Le*, *FractalNet* и *SMASH*.
- **Микро поиск** - это подход, при котором контроллер проектирует модули или строительные блоки, которые объединяются при построении конечной сети. Работы, которые реализуют этот подход: *Progressive NAS (PNAS)*, *NASNet*.

<https://arxiv.org/abs/1611.01578>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “***Efficient Neural Architecture Search (ENAS) via Parameter Sharing***”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Макро поиск

Controller

Child Model

input

DAG

При Макро-поиске контроллер принимает **2 решения** для каждого слоя в дочерней модели:

- Выбор выполняемой операции на. Для контроллера доступно 6 видов операций:
  - convolutions with filter sizes  $3 \times 3$  and  $5 \times 5$ ;
  - depthwise-separable convolutions with filter sizes  $3 \times 3$  and  $5 \times 5$ ;
  - max pooling and average;
  - pooling of kernel size  $3 \times 3$ .
- Выбор слоя для соединения или для пропуска соединения.

<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “***Efficient Neural Architecture Search (ENAS) via Parameter Sharing***”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Макро поиск

Controller

Child Model

input

DAG

- Сверточный слой 1 (красный)
  - Первый временной шаг контроллера. Выход этого временного шага **softmax** для получения вектора вероятностей, который переводится в операцию **conv3x3**.



<https://arxiv.org/abs/1611.01578>

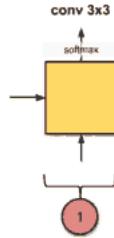
<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “***Efficient Neural Architecture Search (ENAS) via Parameter Sharing***”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Макро поиск

Controller



Child Model



DAG



- Сверточный слой 1 (красный). Первый временной шаг контроллера.
  - 1. Выход этого временного шага - **softmax** для получения вектора вероятностей, который переводится в операцию **conv3×3**.
- Сверточный слой 2 (зеленый).
  - 1. Выбран слой для подключения №1.
  - 2. Сгенерирована операция **sep5×5**.

\* Для дочерней модели это означает, что мы сначала выполняем **sep5×5** операцию на выходе предыдущего слоя. Затем этот выход объединяется по глубине вместе с выходом **слоя 1**, то есть выходом из красного слоя.

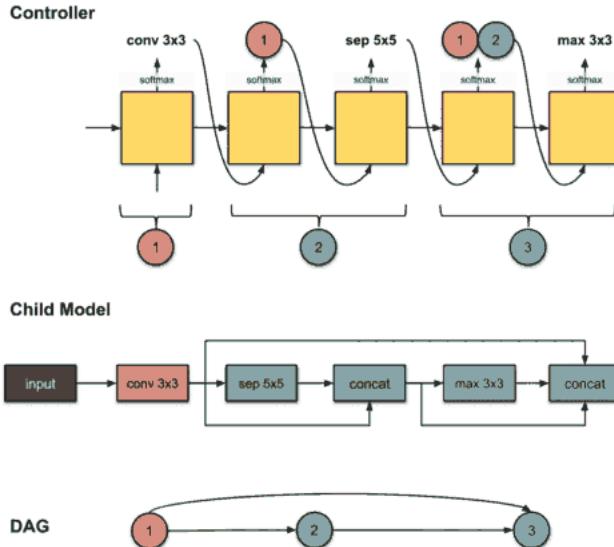
<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “**Efficient Neural Architecture Search (ENAS) via Parameter Sharing**”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Макро поиск



- **Сврточный слой 1 (красный).** Первый временной шаг контроллера.
  - 1. Выход этого временного шага - **softmax** для получения вектора вероятностей, который переводится в операцию **conv3×3**.
- **Сврточный слой 2 (зеленый).**
  - 1. Выбран слой для подключения №1.
  - 2. Сгенерирована операция **sep5×5**.
- **Сврточный слой 3 (синий)**
  - 1. Выбраны слои для подключения №1 и №2.
  - 2. Сгенерирована операция **max3×3**.
- **Сврточный слой 4 (фиолетовый)**
  - 1. Выбран слой для подключения №1^3.
  - 2. Сгенерирована операция **conv5×5**.

<https://arxiv.org/abs/1611.01578>

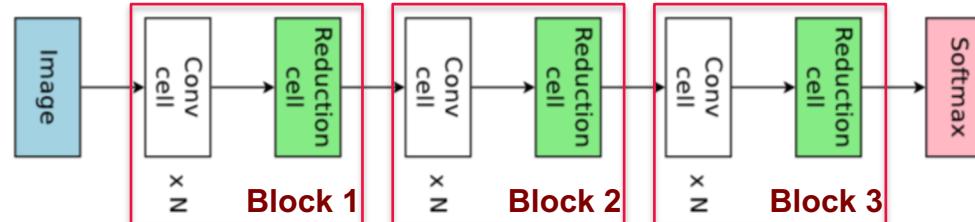
<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “***Efficient Neural Architecture Search (ENAS) via Parameter Sharing***”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск

- Микро-поиск проектирует только один строительный блок, архитектура которого повторяется в окончательной архитектуре. ENAS может генерировать 2 вида строительных блоков: **сверточную ячейку (convolutional cell)** и **ячейку уменьшения размерности (reduction cell)**. Оба типа схожи, единственное отличие в **reduction cell** состоит в том, что операции применяются с шагом 2, что уменьшает пространственные измерения.
- Идея микро-поиска состоит в том, чтобы **создать единую архитектуру для сверточной ячейки и повторить эту же архитектуру в окончательной модели**. В приведенном ниже примере окончательная модель состоит из сверточной ячейки, архитектура которой повторяется  **$N$  раз в 3 блоках**. Как упомянуто в предыдущем абзаце, ячейка сокращения аналогична архитектуре сверточной ячейки.



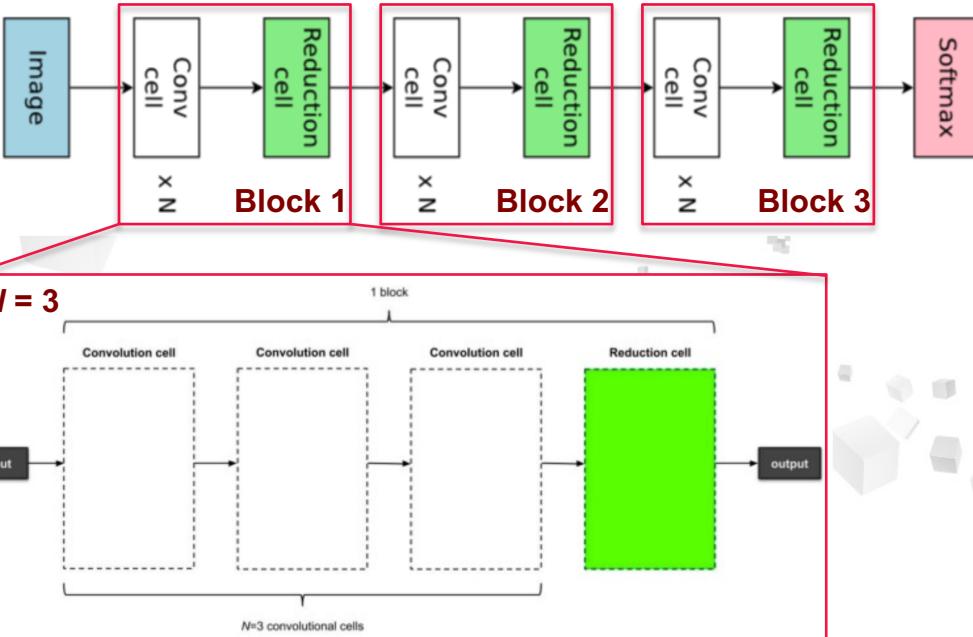
<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “**Efficient Neural Architecture Search (ENAS) via Parameter Sharing**”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск. Иерархия.



- В структуре сетей, полученных с помощью стратегии **микро-поиска** установлена **жесткая иерархия строительных единиц** (от самого сложного к самому простому):
  - **Блок.**
  - **Сверточная ячейка / ячейка уменьшения размерности.**
  - **Узел.**
- Дочерняя модель состоит из **нескольких блоков**.
- Каждый блок состоит из  **$N$  сверточных ячеек** и **1 ячейки уменьшения размерности**.
- Каждая **сверточная / “уменьшающая” ячейка** содержит  **$B$  узлов**.
- И каждый **узел** состоит из **стандартных сверточных операций** (мы увидим это позже).
- **$N$  и  $B$**  - это гиперпараметры, которые может настраивать архитектор.

<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “*Efficient Neural Architecture Search (ENAS) via Parameter Sharing*”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск. Строение сверточной ячейки.

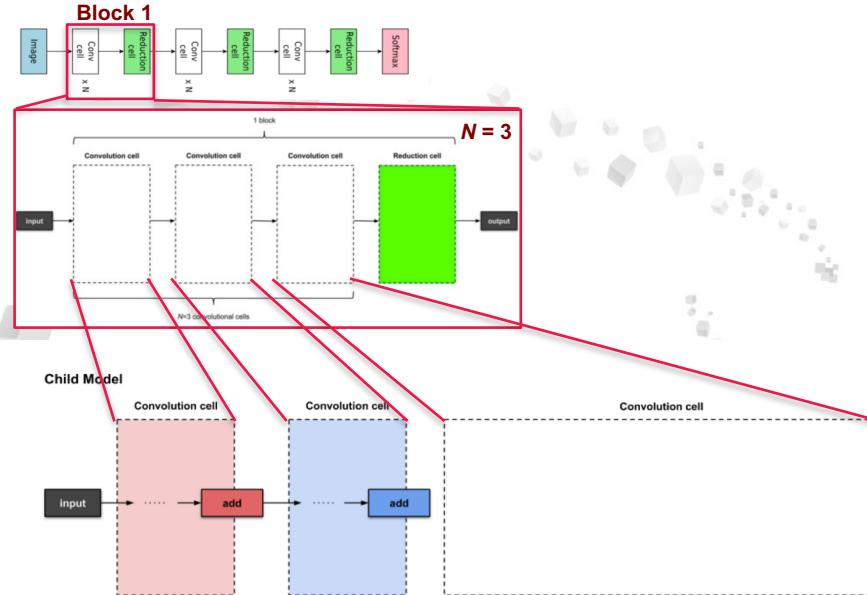


Рис. 1.2.5: «Подготовка» третьей сверточной ячейки в микропоиске.

<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

- В структуре сетей, полученных с помощью стратегии **микро-поиска** установлена **жесткая иерархия строительных единиц** (от самого сложного к самому простому):
  - **Блок.**
  - **Сверточная ячейка / ячейка уменьшения размерности.**
  - **Узел.**
- Дочерняя модель состоит из **нескольких блоков**.
- Каждый блок состоит из  **$N$  сверточных ячеек** и 1 ячейки уменьшения размерности.
- Каждая сверточная / “уменьшающая” ячейка содержит  **$B$  узлов**.
- И каждый узел состоит из **стандартных сверточных операций** (мы увидим это позже).
- **$N$  и  $B$**  - это гиперпараметры, которые может настраивать архитектор.

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “*Efficient Neural Architecture Search (ENAS) via Parameter Sharing*”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск. Строение сверточной ячейки.

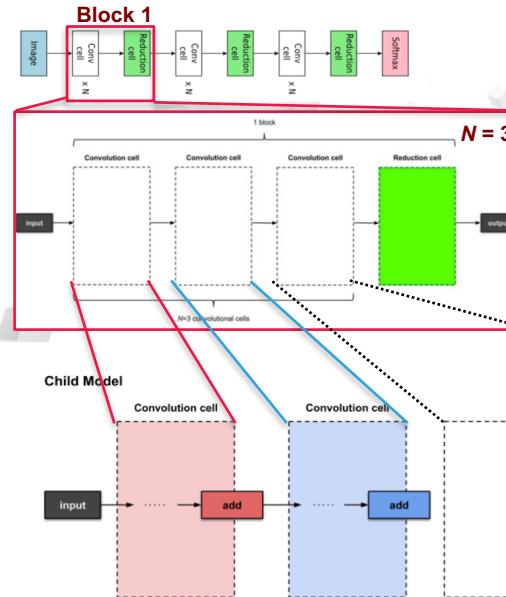


Рис. 1.2.5: «Подготовка» третьей сверточной ячейки в микропоиске.

- Создадим дочернюю модель, которая имеет **1 блок**. Этот блок содержит  $N = 3$  сверточных ячейки и 1 ячейку сокращения размерности, и каждая ячейка содержит  $B = 4$  узла.
- В предположим, что у нас уже есть **две сверточные ячейки (из 4)**. Обратите внимание, что последние операции каждой из этих двух ячеек являются **add** операциями.
- Для простоты **две первые ячейки (красная и синяя)** будут рассматриваться в качестве **1 и 2 узлов соответственно**. Два других узла (зеленый и фиолетовый) попадают в сверточную ячейку, которую мы сейчас строим.

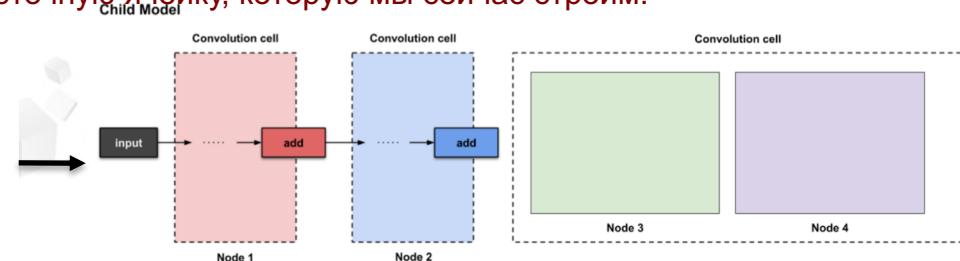


Рис. 1.2.6: Идентификация 4 узлов при построении сверточной ячейки № 3.

<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

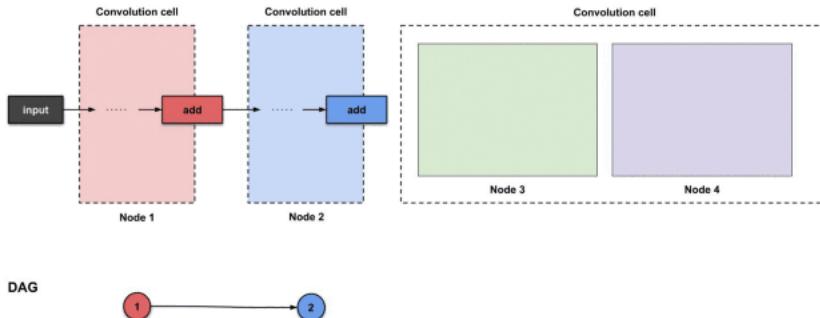
# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “***Efficient Neural Architecture Search (ENAS) via Parameter Sharing***”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск. Строение сверточной ячейки.

Controller

Child Model



- Начиная с этого места, можно спокойно игнорировать метки «Сверточная ячейка», которые приведены на предыдущих рисунках, и концентрироваться на метках «Узлы»:
  - Узел 1 – красный.
  - Узел 2 – синий.
  - Узел 3 – зеленый.
  - Узел 4 – фиолетовый.
- Узел 3 - это то место, где начинается наша конструкция. Контроллер генерирует **4 решения (точнее 2 набора решений)**:
  - 2 узла для соединения с текущим;
  - 2 операции для выполнения на соответствующих подключаемых узлах.

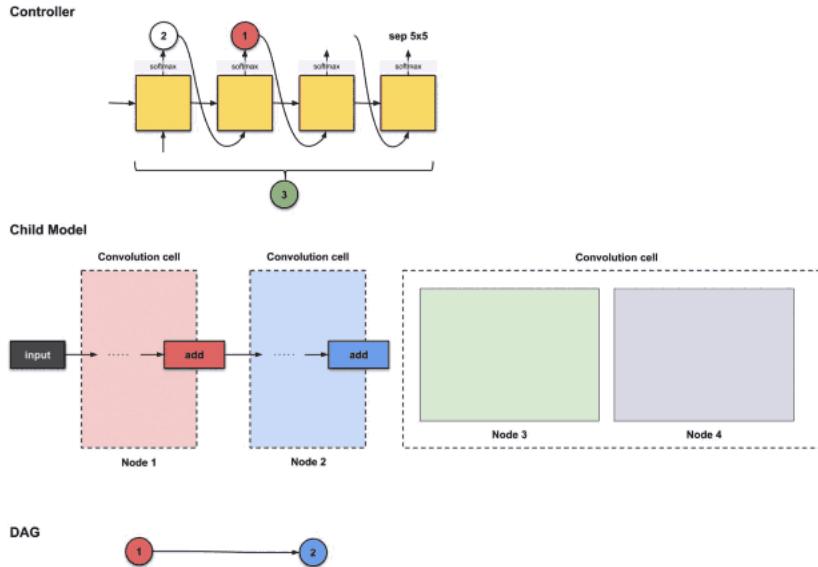
<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “***Efficient Neural Architecture Search (ENAS) via Parameter Sharing***”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск. Строение сверточной ячейки.



- Начиная с этого места, можно спокойно игнорировать метки «Сверточная ячейка», которые приведены на предыдущих рисунках, и концентрироваться на метках «Узлы»:
  - Узел 1 – красный.
  - Узел 2 – синий.
  - Узел 3 – зеленый.
  - Узел 4 – фиолетовый.
- Узел 3 - это то место, где начинается наша конструкция. Контроллер генерирует **4 решения (точнее 2 набора решений)**:
  - 2 узла для соединения с текущим;
  - 2 операции для выполнения на соответствующих подключаемых узлах.
- В нашем случае контроллер выбрал:
  - Узел 2, Узел 1.
  - И соответствующие операции **avg5×5** и **sep5×5** на каждом из четырех шагов по времени.

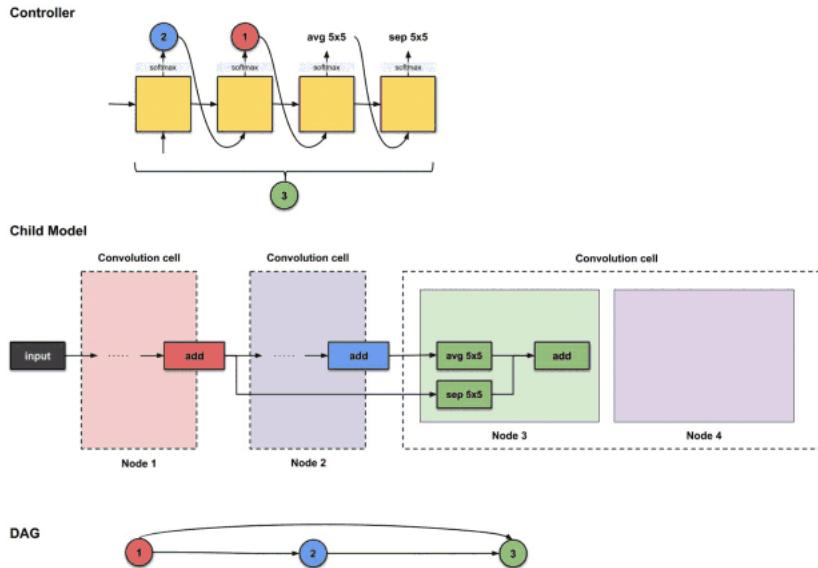
<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “**Efficient Neural Architecture Search (ENAS) via Parameter Sharing**”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск. Строение сверточной ячейки.



- Узел 3 - это то место, где начинается наша конструкция.  
Контроллер генерирует **4 решения** (точнее **2 набора решений**):
  - 2 узла для соединения с текущим;
  - 2 операции для выполнения на соответствующих подключаемых узлах.
- В нашем случае контроллер выбрал:
  - Узел 2, Узел 1.
  - И соответствующие операции **avg5×5** и **sep5×5** на каждом из четырех шагов по времени.
- Теперь для Узла 4 контроллер повторяем те же шаги, но теперь у него есть три узла на выбор для потенциального соединения (Узлы 1, 2 и 3).
- Узел 4. Сгенерировано:
  - Узел 3, Узел 1.
  - **id** и **avg3×3**

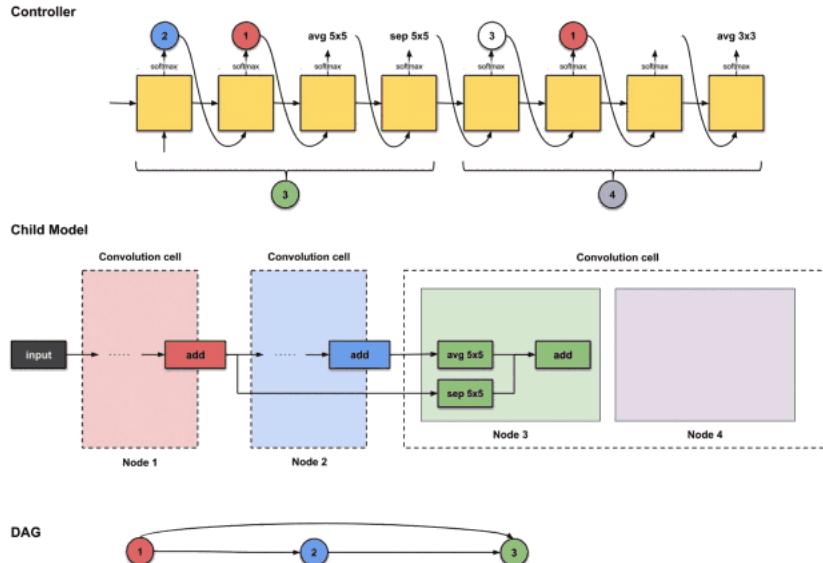
<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “***Efficient Neural Architecture Search (ENAS) via Parameter Sharing***”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск. Строение сверточной ячейки.



- Узел 3 - это то место, где начинается наша конструкция.  
Контроллер генерирует **4 решения** (точнее **2 набора решений**):
  - 2 узла для соединения с текущим;
  - 2 операции для выполнения на соответствующих подключаемых узлах.
- В нашем случае контроллер выбрал:
  - Узел 2, Узел 1.
  - И соответствующие операции **avg5×5** и **sep5×5** на каждом из четырех шагов по времени.
- Теперь для Узла 4 контроллер повторяем те же шаги, но теперь у него есть три узла на выбор для потенциального соединения (Узлы 1, 2 и 3).
- Узел 4. Сгенерировано:
  - Узел 3, Узел 1.
  - **id** и **avg3×3**
- После этого структура меняется следующим образом:

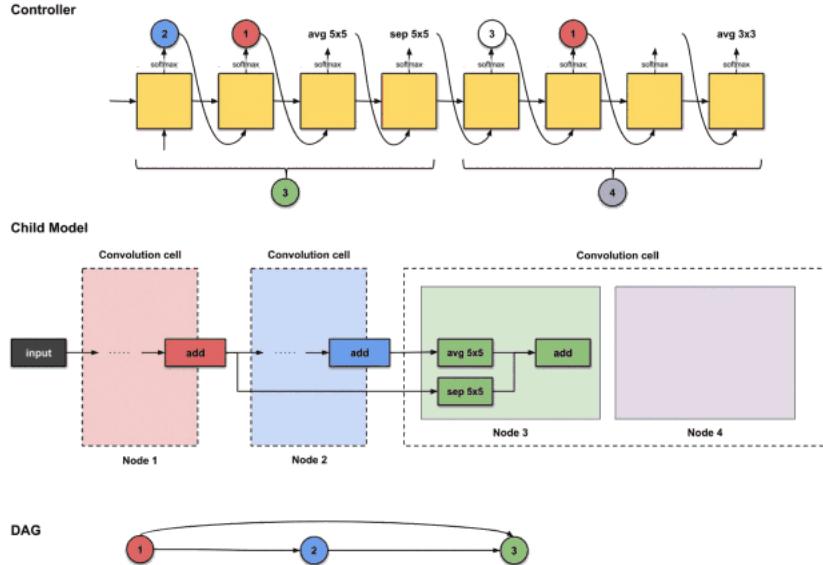
<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

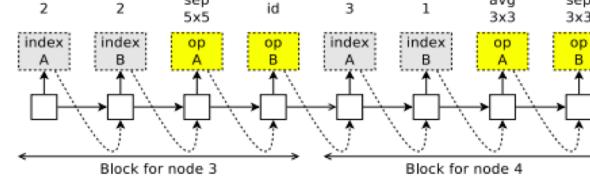
# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “**Efficient Neural Architecture Search (ENAS) via Parameter Sharing**”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

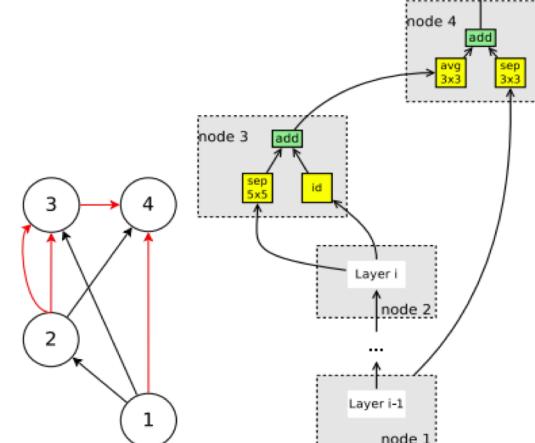
## Микро поиск. Строение сверточной ячейки.



- Сверточная ячейка сгенерирована.



Layer i+1



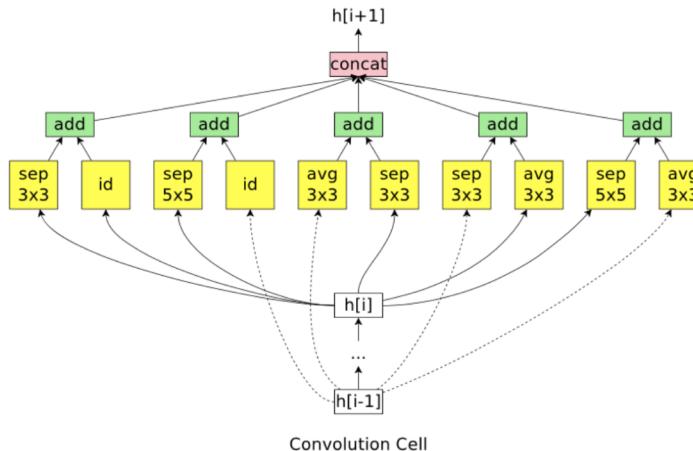
<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “***Efficient Neural Architecture Search (ENAS) via Parameter Sharing***”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск. Строение сверточной ячейки.



- Сверточная ячейка сгенерирована.
- **Все сверточные ячейки в конечной дочерней модели будут иметь одинаковую архитектуру.** Точно так же все ячейки сокращения размерности (архитектура которых отличается от структуры сверточной ячейки наличием операций шага 2) в конечной дочерней модели будут использовать одну и ту же архитектуру.

Рис. 1.2.11. Пример ячейки свертки (с 7 узлами), обнаруженной в пространстве микропоиска (кредиты).

<https://arxiv.org/abs/1611.01578>

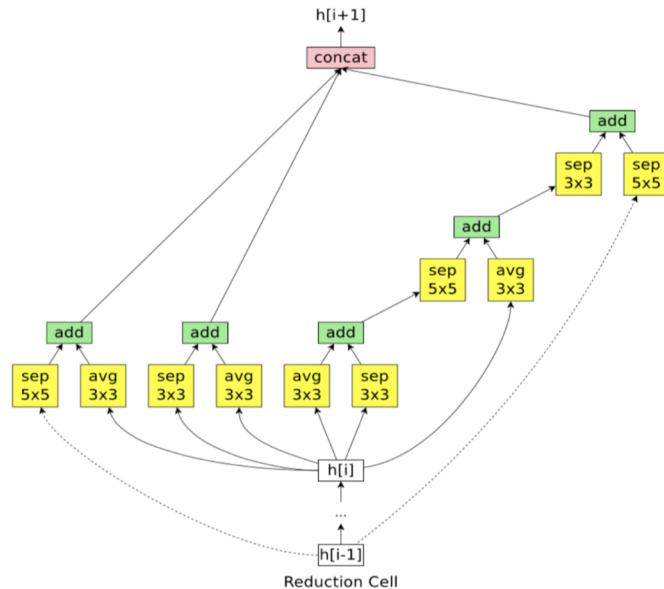
<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

HUAWEI TECHNOLOGIES CO., LTD.

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “**Efficient Neural Architecture Search (ENAS) via Parameter Sharing**”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

## Микро поиск. Строение сверточной ячейки.



- Сверточная ячейка сгенерирована.
- **Все сверточные ячейки в конечной дочерней модели будут иметь одинаковую архитектуру.** Точно так же все ячейки сокращения размерности (архитектура которых отличается от структуры сверточной ячейки наличием операций шага 2) в конечной дочерней модели будут использовать одну и ту же архитектуру.

Рис. 1.2.12: Пример ячейки редукции (с 7 узлами), обнаруженной в пространстве микропоиска ( кредиты ).

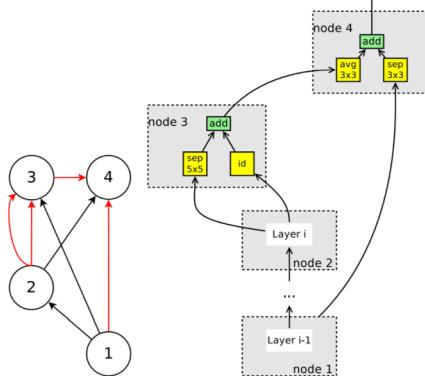
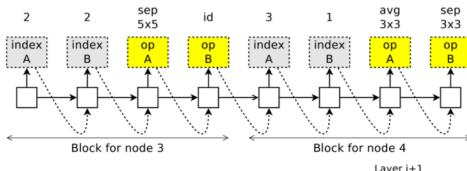
<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

HUAWEI TECHNOLOGIES CO., LTD.

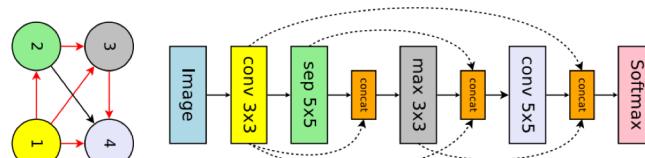
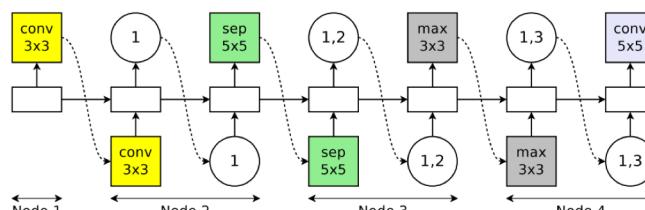
# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “**Efficient Neural Architecture Search (ENAS)** via Parameter Sharing”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.



- Сверточная ячейка сгенерирована.

- Что такого «эффективного» в ENAS? Ответ: трансферное обучение. Если обучение между двумя узлами было выполнено ранее, весовые коэффициенты от сверточных фильтров и сверток  $1 \times 1$  (для сохранения количества выходов канала; не упомянутых в предыдущих разделах) будут использоваться повторно. Это то, что делает ENAS быстрее, чем его предшественники! (с 800 гру часов до 32!)



- Авторы заявляют, что контроллер выберет решение, при котором skip-connections не требуется.

Figure 5. An example run of the controller for our search space over convolutional cells. Top: the controller's outputs. In our search space for convolutional cells, node 1 and node 2 are the cell's inputs, so the controller only has to design node 3 and node 4. Bottom Left: The corresponding DAG, where red edges represent the activated connections. Bottom Right: the convolutional cell according to the controller's sample.

<https://arxiv.org/abs/1611.01578>

<https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>

HUAWEI TECHNOLOGIES CO., LTD.

# Развитие идеи NAS

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean “**Efficient Neural Architecture Search (ENAS) via Parameter Sharing**”  
(Google Brain Language Technology Institute, Carnegie Mellon University, Department of Computer Science, Stanford University) 2018.

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	<b>2.56</b>
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	<b>6.92</b>
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	<b>3.65</b>
ENAS + macro search space	1	0.32	21.3	4.23
ENAS + macro search space + more channels	1	0.32	38.0	<b>3.87</b>
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	<b>2.65</b>
ENAS + micro search space	1	0.45	4.6	3.54
ENAS + micro search space + CutOut	1	0.45	4.6	<b>2.89</b>

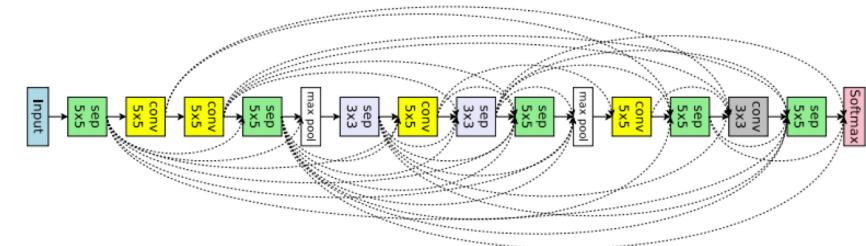
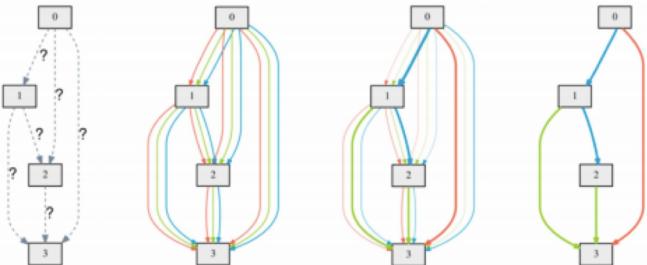


Figure 7. ENAS’s discovered network from the macro search space for image classification.

Результаты сравнения самых удачных решений в 2018 году для набора данных CIFAR10.

# Развитие идеи NAS

- Применение градиентной оптимизации для поиска архитектур *Hanxiao Liu, Karen Simonyan, Yiming Yang “DARTS: Differentiable Architecture Search” (Carnegie Mellon UniversityGoogle) 2018.*



DARTS (Liu et al., 2018)

Find encoding weights  $\alpha^*$  that

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

Learn design of normal and reduction cells

Подробное знакомство с методом и его практическое использование 18.08.2020

Лектор:

старший инженер Huawei  
Власов Роман

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  $\alpha$  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  $\nabla_w \mathcal{L}_{train}(w, \alpha)$

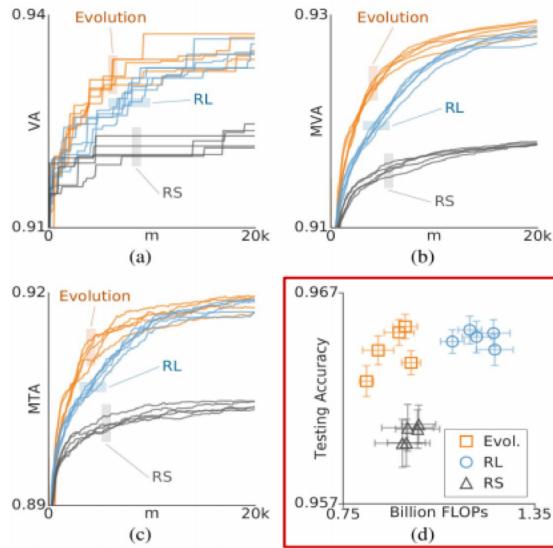
Derive the final architecture based on the learned  $\alpha$ .

# Развитие идеи NAS

Очень интересная статистика и идеи на тему “**Лучше ли весь пул самых современных, продвинутых NAS подходов для исследования пространства архитектур, чем Random Search?**” в материале **Nikhil Naik 2019 года.**

[https://metalearning-cvpr2019.github.io/assets/CVPR\\_2019\\_Metalearning\\_Tutorial\\_Nikhil\\_Naik.pdf](https://metalearning-cvpr2019.github.io/assets/CVPR_2019_Metalearning_Tutorial_Nikhil_Naik.pdf)

## Are Intelligent Search Strategies Better Than Random Search?



**Real et al. (2018)**  
The difference in accuracy  
between best models  
found by random search,  
RL, and Evolution is  
**less than 1%** on CIFAR-10

[https://metalearning-cvpr2019.github.io/assets/CVPR\\_2019\\_Metalearning\\_Tutorial\\_Nikhil\\_Naik.pdf](https://metalearning-cvpr2019.github.io/assets/CVPR_2019_Metalearning_Tutorial_Nikhil_Naik.pdf)

# Развитие идеи NAS

“Лучше ли весь пул самых современных, продвинутых NAS подходов для исследования пространства архитектур, чем Random Search?” в материале *Nikhil Naik*.

## Are Intelligent Search Strategies Better Than Random Search?

Architecture	Source	Test Error		Params (M)
		Best	Average	
NASNet-A <sup>**</sup>	[52]	N/A	2.65	3.3
AmoebaNet-B <sup>*</sup>	[43]	N/A	2.55 ± 0.05	2.8
ProxylessNAS <sup>†</sup>	[7]	2.08	N/A	5.7
GHN <sup>#†</sup>	[50]	N/A	2.84 ± 0.07	5.7
SNAS <sup>†</sup>	[47]	N/A	2.85 ± 0.02	2.8
ENAS <sup>†</sup>	[41]	2.89	N/A	4.6
ENAS	[34]	2.91	N/A	4.2
Random search baseline	[34]	N/A	3.29 ± 0.15	3.2
DARTS (first order)	[34]	N/A	3.00 ± 0.14	3.3
DARTS (second order)	[34]	N/A	2.76 ± 0.09	3.3
DARTS (second order) <sup>‡</sup>	Ours	2.62	2.78 ± 0.12	3.3
ASHA baseline	Ours	2.85	3.03 ± 0.13	2.2
Random search WS <sup>‡</sup>	Ours	2.71	2.85 ± 0.08	4.3

**Li and Talwalkar (2019)**  
Random search baseline finds a model with 2.71% error on CIFAR-10, comparable to best NAS methods based on RL, Evolution, Gradient Descent

## Are Intelligent Search Strategies Better Than Random Search?



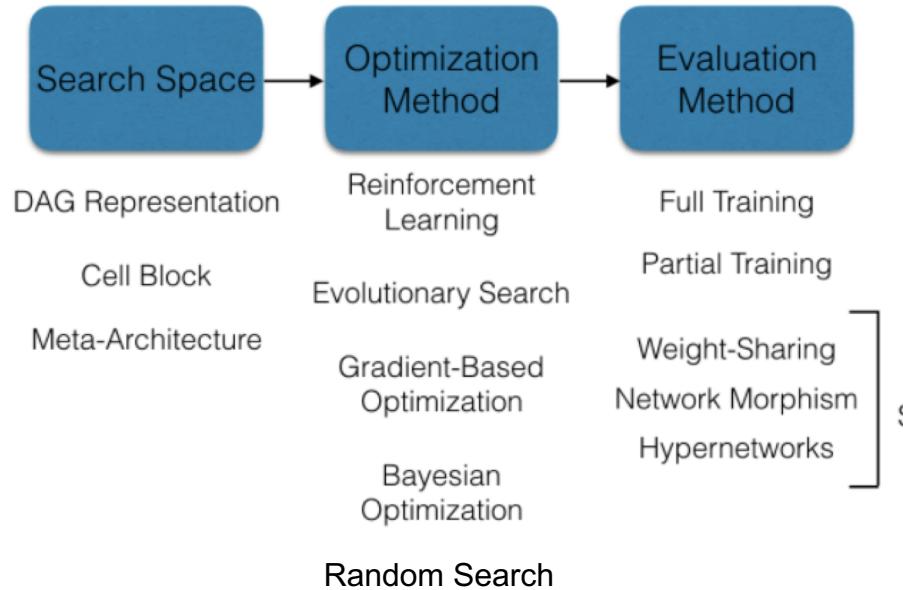
**Xie et al. (2019)**

A network consisting of multiple randomly wired “cells” is only **1.3% less accurate** than a similar capacity NAS models on **ImageNet**

network	test size	epochs	top-1 acc.	top-5 acc.	FLOPs (B)	params (M)
NASNet-A [56]	$331^2$	>250	82.7	96.2	23.8	88.9
Amoeba-B [34]	$331^2$	>250	82.3	96.1	22.3	84.0
Amoeba-A [34]	$331^2$	>250	82.8	96.1	23.1	86.7
PNASNet-5 [26]	$331^2$	>250	82.9	96.2	25.0	86.1
<b>RandWire-WS</b>	$320^2$	100	$81.6 \pm 0.13$	$95.6 \pm 0.07$	$16.0 \pm 0.36$	$61.5 \pm 1.32$

# Развитие идеи NAS

## Components of NAS



- Основные компоненты NAS представлены на данном слайде.

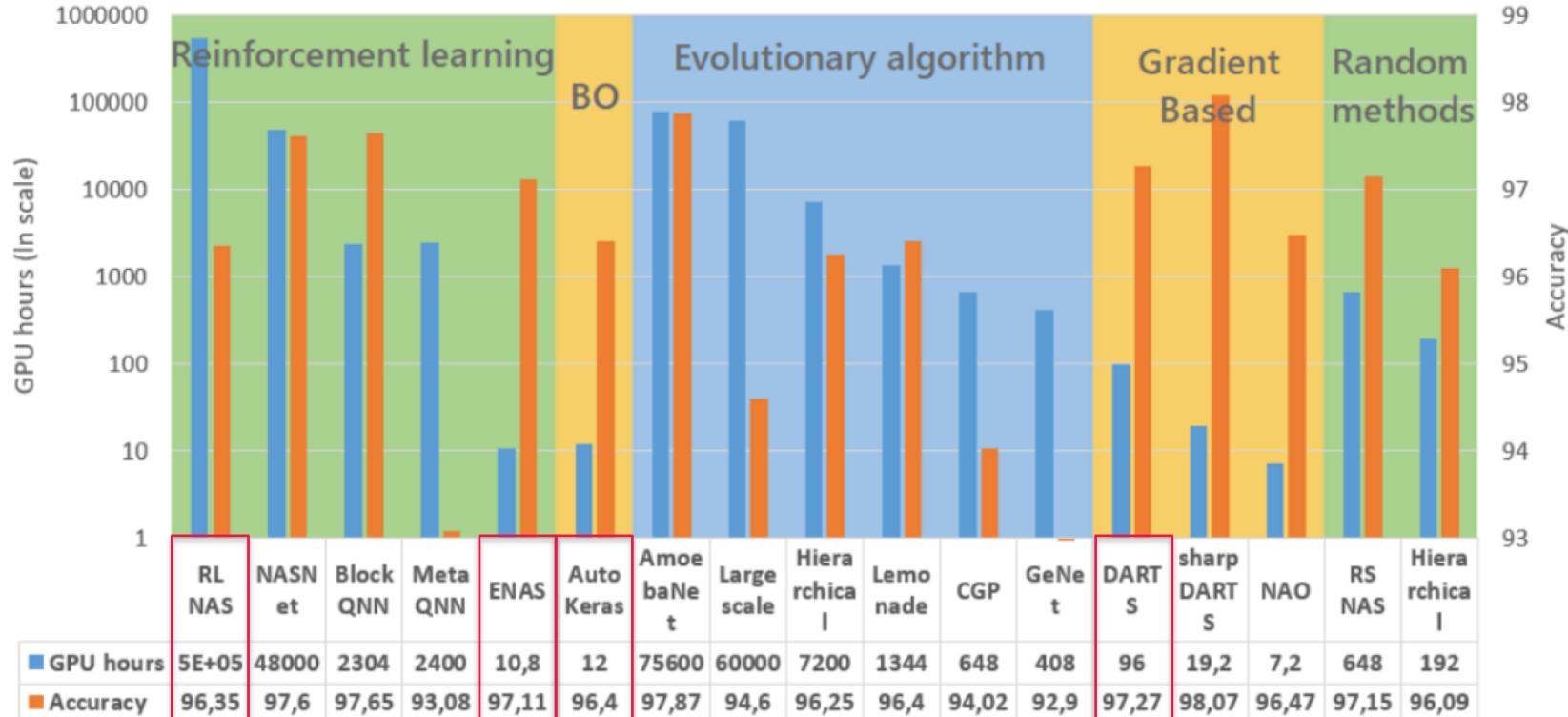
- Современный спектр алгоритмов NAS огромен, но, в подавляющем большинстве, все они укладываются в приведенную схему и, используя различные комбинации приведенных блоков, дают хорошие результаты для разных задач.

- Некоторые наиболее известные работы NAS:

- Reinforcement learning**  
[Neural Architecture Search with Reinforcement Learning](#) (Zoph and Le, 2016)  
[NASNet](#) (Zoph et al., 2017)  
[ENAS](#) (Pham et al., 2018)
- Evolutionary algorithm**  
[Hierarchical Evo](#) (Liu et al., 2017)  
[AmoebaNet](#) (Real et al., 2018)
- Sequential model-based optimization (SMBO)**  
[PNAS](#) (Liu et al., 2017)
- Bayesian optimisation**  
[Auto-Keras](#) (Jin et al., 2018)  
[NASBOT](#) (Kandasamy et al. 2018)
- Gradient-based optimization**  
[SNAS](#) (Xie et al., 2018)  
[DARTS](#) (Liu et al., 2018)

<https://www.houseofbots.com/news-detail/4440-1-how-does-neural-architecture-search-work>

# Развитие идеи NAS



Результаты сравнения самых удачных NAS решений на 31 октября 2019 года для набора данных CIFAR10.

<https://blog.ml6.eu/automated-machine-learning-using-auto-keras-f9b7002aee0e>

# Развитие идеи NAS

## CIFAR-10 ↗

Classify 32x32 colour images into 10 categories.

Method	(expand all   collapse all)	Accuracy (%)
✗ ⓘ Big Transfer (BiT): General Visual Representation Learning	(Dec 2019)	99.37%
✗ ⓘ GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism	(Nov 2018, arXiv 2018)	99.00%
✗ ⓘ EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks	(May 2019, arXiv 2019)	98.90%
Mingxing Tan, Quoc V. Le		
Convolutional Neural Networks (ConvNets) are commonly developed at a fixed resource budget, and then scaled up for better accuracy if more resources are available. In this paper, we systematically study model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Based on this observation, we propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective compound coefficient. We demonstrate the effectiveness of this method on scaling up MobileNets and ResNet. To go even further, we use neural architecture search to design a new baseline network and scale it up to obtain a family of models, called EfficientNets, which achieve much better accuracy and efficiency than previous ConvNets. In particular, our EfficientNet-B7 achieves state-of-the-art 84.4% top-1 / 97.1% top-5 accuracy on ImageNet, while being 8.4x smaller and 6.1x faster on inference than the best existing ConvNet. Our EfficientNets also transfer well and achieve state-of-the-art accuracy on CIFAR-100 (91.7%), Flowers (98.8%), and 3 other transfer learning datasets, with an order of magnitude fewer parameters. Source code is at <a href="https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet">https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet</a> .		
✗ ⓘ A Survey on Neural Architecture Search	(May 2019, arXiv 2019)	98.67%
Martin Wistuba, Ambrish Rawat, Tejaswini Pedapati		
The growing interest in both the automation of machine learning and deep learning has inevitably led to the development of automated methods for neural architecture optimization. The choice of the network architecture has proven to be critical, and many advances in deep learning spring from its immediate improvements. However, deep learning techniques are computationally intensive and their application requires a high level of domain knowledge. Therefore, even partial automation of this process would help make deep learning more accessible to both researchers and practitioners. With this survey, we provide a formalism which unifies and categorizes the landscape of existing methods along with a detailed analysis that compares and contrasts the different approaches. We achieve this via a discussion of common architecture search spaces and architecture optimization algorithms based on principles of reinforcement learning and evolutionary algorithms along with approaches that incorporate surrogate and one-shot models. Additionally, we address the new research directions which include constrained and multi-objective architecture search as well as automated data augmentation, optimizer and activation function search.		
✗ ⓘ AutoAugment: Learning Augmentation Policies from Data	(May 2018, arXiv 2018)	98.52%
✗ ⓘ XNAS: Neural Architecture Search with Expert Advice	(Jun 2019, arXiv 2019)	98.40%
Niv Nayman, Asaf Noy, Tal Ridnik, Itamar Friedman, Rong Jin, Lihai Zelnik-Manor		
This paper introduces a novel optimization method for differential neural architecture search, based on the theory of prediction with expert advice. Its optimization criterion is well fitted for an architecture-selection, i.e., it minimizes the regret incurred by a sub-optimal selection of operations. Unlike previous search relaxations, that require hard pruning of architectures, our method is designed to dynamically wipe out inferior architectures and enhance superior ones. It achieves an optimal worst-case regret bound and suggests the use of multiple learning-rates, based on the amount of information carried by the backward gradients. Experiments show that our algorithm achieves a strong performance over several image classification datasets. Specifically, it obtains an error rate of 1.6% for CIFAR-10, 24% for ImageNet under mobile settings, and achieves state-of-the-art results on three additional datasets.		
✗ ⓘ ShakeDrop Regularization	(Feb 2018, ICLR 2018)	97.69%
✗ ⓘ Improved Regularization of Convolutional Neural Networks with Cutout	(Aug 2017, arXiv 2017)	97.44%
✗ ⓘ Random Erasing Data Augmentation	(Aug 2017, arXiv 2017)	96.92%
✗ ⓘ Drop-Activation: Implicit Parameter Reduction and Harmonic Regularization	(Nov 2018, arXiv 2018)	96.55%
✗ ⓘ Densely Connected Convolutional Networks	(Aug 2016, arXiv 2016)	96.54%
✗ ⓘ Fractional Max-Pooling	(Dec 2014)	96.53%
✗ ⓘ Residual Networks of Residual Networks: Multilevel Residual Networks	(Aug 2016, arXiv 2017)	96.23%

Результаты сравнения применения самых удачных NAS решений на 17 августа 2020 года для набора данных CIFAR10.

<https://benchmarks.ai/cifar-10>

# Развитие идеи NAS

Ресурсы, агрегирующие всю текущую информацию в области NAS.



Search

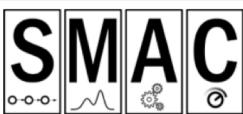
Follow Us [Twitter](#) [Email](#)

AutoML Freiburg-Hannover

[Home](#) [Blog](#) [AutoML](#) [AAD](#) [Analysis](#) [Events](#) [Book](#) [Jobs](#) [Team & Partners](#)

## AutoML ...

provides methods and processes to make Machine Learning available for non-Machine Learning experts, to improve efficiency of Machine Learning and to accelerate research on Machine Learning. Machine learning (ML) has achieved considerable successes in recent years and an ever-growing number of disciplines rely on it. However, this success crucially relies on human machine learning experts to perform manual tasks. As the complexity of these tasks is often beyond non-ML-experts, the rapid growth of machine learning applications has created a demand for off-the-shelf machine learning methods that can be used easily and without expert knowledge. We call the resulting research area that targets progressive automation of machine learning AutoML.



SMAC  
Sequential Model-based Algorithm Configuration is a state-of-the-art tool to



Auto-Sklearn



BOHB

## LITERATURE ON NEURAL ARCHITECTURE SEARCH

Maintained by [Difan Deng](#) and [Marius Lindauer](#); Last update: July 27th 2020

The following list considers papers related to neural architecture search. It is by no means complete. We highlight papers accepted at conferences and journals; this should hopefully provide some guidance towards high-quality papers. If you miss a paper on the list, please let us know.

**Update [Sept'19]:** Although NAS methods steadily improve, the quality of empirical evaluations in this field are still lagging behind compared to other areas in machine learning, AI and optimization. We would therefore like to share some best practices for empirical evaluations of NAS methods, which we believe will facilitate sustained and measurable progress in the field. If you are interested in a teaser, please read our [blog post](#) or directly jump to our [checklist](#).

- Weight-Sharing Neural Architecture Search: A Battle to Shrink the Optimization Gap (Xie et al. 2020)  
<https://arxiv.org/abs/2008.01475>
- MCUNet: Tiny Deep Learning on IoT Devices (Lin et al. 2020)  
<https://arxiv.org/abs/2007.10319>
- Search What You Want: Barrier Panely NAS for Mixed Precision Quantization (Yu et al. 2020; accepted at ECCV 2020)  
<https://arxiv.org/abs/2007.10026>
- NSGANetV2: Evolutionary Multi-Objective Surrogate-Assisted Neural Architecture Search (Lu et al. 2020; accepted at ECCV 2020)  
<https://arxiv.org/abs/2007.10396>
- CATCH: Context-based Meta Reinforcement Learning for Transferrable Architecture Search (Chen et al. 2020; accepted at ECCV 2020)  
<https://arxiv.org/abs/2007.09380>
- Standing on the Shoulders of Giants: Hardware and Neural Architecture Co-Search with Hot Start (Jiang et al. 2020; accepted at IEEE Transactions On Computer-Aided Design of Integrated Circuits and System)  
<https://arxiv.org/abs/2007.09087>
- Off-Policy Reinforcement Learning for Efficient and Effective GAN Architecture Search (Tian et al. 2020; accepted at ECCV 2020)  
<https://arxiv.org/abs/2007.09180>
- Neural Architecture Search for Speech Recognition (Hu et al. 2020)  
<https://arxiv.org/abs/2007.08818>
- BRP-NAS: Prediction-based NAS using GCNs (Chau et al. 2020)  
<https://arxiv.org/abs/2007.08668>
- Finding Non-Uniform Quantization Schemes using Multi-Task Gaussian Processes (do Nascimento et al. 2020; accepted at ECCV 2020)  
<https://ui.adsabs.harvard.edu/abs/2020arXiv200707743G/abstract>
- One-Shot Neural Architecture Search via Novelty Driven Sampling (Zhang et al. 2020; accepted at IJCAI 2020)  
<https://www.ijcai.org/Proceedings/2020/0441.pdf>

<https://www.automl.org/>

HUAWEI TECHNOLOGIES CO., LTD.

# Развитие идеи NAS

Ресурсы, агрегирующие всю текущую информацию в области NAS.

D-X-Y / Awesome-AutoDL

Code Issues Pull requests Actions Projects Wiki Security Insights

master 1 branch 0 tags Go to file Add file Code

9b34d5b 7 days ago 117 commits

LICENSE updates 9 months ago

README.md Update MiLeNAS 7 days ago

README.md

**Awesome AutoDL** awesome

A curated list of automated deep learning related resources. Inspired by [awesome-deep-vision](#), [awesome-adversarial-machine-learning](#), [awesome-deep-learning-papers](#), and [awesome-architecture-search](#).

Please feel free to [pull requests](#) or [open an issue](#) to add papers.

**Table of Contents**

- Awesome Blogs
- Awesome AutoDL Libraies
- Deep Learning-based NAS and HPO
  - 2020 Venues
  - 2019 Venues
  - 2018 Venues
  - 2017 Venues
  - Previous Venues
  - arXiv
- Awesome Surveys

<https://github.com/D-X-Y/Awesome-AutoDL>

## Deep Learning-based NAS and HPO

Type	G	RL	EA	PD	Other
Explanation	gradient-based	reinforcement learning	evolutionary algorithm	performance prediction	other types

### 2020

Title	Venue	Type	Code
<a href="#">Optimizing Millions of Hyperparameters by Implicit Differentiation</a>	AISTATS	G	-
<a href="#">Evolving Machine Learning Algorithms From Scratch</a>	ICML	-	-
<a href="#">NADS: Neural Architecture Distribution Search for Uncertainty Awareness</a>	ICML	-	-
<a href="#">Generative Teaching Networks: Accelerating Neural Architecture Search by Learning to Generate Synthetic Training Data</a>	ICML	-	-
<a href="#">Neural Architecture Search in a Proxy Validation Loss Landscape</a>	ICML	-	-
<a href="#">MiLeNAS: Efficient Neural Architecture Search via Mixed-Level Reformulation</a>	CVPR	G	<a href="#">GitHub</a>
<a href="#">Binarizing MobileNet via Evolution-based Searching</a>	CVPR	EA	-
<a href="#">Rethinking Performance Estimation in Neural Architecture Search</a>	CVPR	-	<a href="#">GitHub</a>
<a href="#">APQ: Joint Search for Network Architecture, Pruning and Quantization Policy</a>	CVPR	G	<a href="#">GitHub</a>
<a href="#">SGAS: Sequential Greedy Architecture Search</a>	CVPR	G	<a href="#">GitHub</a>
<a href="#">Can Weight Sharing Outperform Random Architecture Search? An Investigation With TuNAS</a>	CVPR	RL	-
<a href="#">FBNV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions</a>	CVPR	G	<a href="#">GitHub</a>
<a href="#">AdversarialNAS: Adversarial Neural Architecture Search for GANs</a>	CVPR	G	<a href="#">GitHub</a>
<a href="#">When NAS Meets Robustness: In Search of Robust Architectures against Adversarial Attacks</a>	CVPR	G	<a href="#">GitHub</a>
<a href="#">Block-wisely Supervised Neural Architecture Search with Knowledge Distillation</a>	CVPR	G	<a href="#">GitHub</a>
<a href="#">Overcoming Multi-Model Forgetting in One-Shot NAS with Diversity Maximization</a>	CVPR	G	<a href="#">GitHub</a>

# Развитие идеи NAS

Основные выводы:

- Современные методы NAS во многом зависят от того, как именно формализуется решаемая задача.
- Формализация задачи зависит от того, как именно разработчик (архитектор) представляет пространство поиска.
- Представление пространства поиска во многом зависит от **типа решаемой задачи**, а следовательно от **данных** (будь то задача классификации изображений или, например, задача DPD).
- Таким образом, заранее зная, какой **набор строительных блоков** будет заложен в **словарь поиска**, по каким **правилам эти блоки будут соединяться**, архитектор заранее очень сильно ограничивает пространство поиска, фактически пользуясь некоторыми **эвристиками и собственным опытом**, полученным в процессе построения моделей для подобных задач.
- **Фактически, единой теории NAS, как таковой не существует.** Существует **набор методов и алгоритмов, основанных на эвристиках исследования графового пространства**.
- Отдельный вопрос касается того, почему **Random Search** действительно так хорош в задачах построения сетей. Если равномерно распределенный поиск хорошо работает в пространстве архитектур следовательно:
  - Весьма вероятно, что относительно “хорошие” минимумы тоже распределены равномерно.
  - “Хорошие” минимумов довольно много, иначе бы рандомный поиск их не нашел бы.
  - Мы что-то до конца не знаем о исследуемом пространстве, что не позволяет нам более качественно исследовать пространство графовых структур и делать это существенно лучше, чем Random Serach.



[www.huawei.com](http://www.huawei.com)

**Copyright © Huawei Technologies Co., Ltd. 2020. All rights reserved.**

All logos and images displayed in this document are the sole property of their respective copyright holders. No endorsement, partnership, or affiliation is suggested or implied. The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.