Auto ML : Automated Machine Learning

Jongwon Kim

포항공과대학교 산업경영학과

Statistics and Data Science Lab.

October 15, 2020

Contents

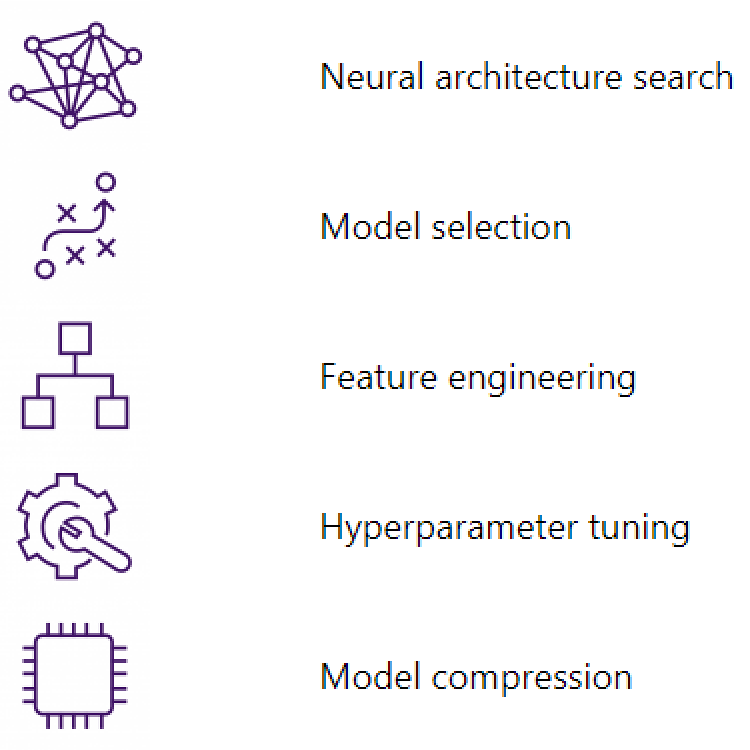
Introduction

- **What is Auto ML?**

Automating the entire process of machine learning is called Auto ML.

In particular, it is mainly used to automate major processes such as Feature Engineering, Neural Architecture Search, and Hyperparamter Optimization.

Synthesizing the process of searching which model to use and which hyperparameter is the optimum solution for each model is called Combined Algorithm Selection and Hyperparameter (CASH).



https://azure.microsoft.com/en-us/services/machine-learning/automatedml/

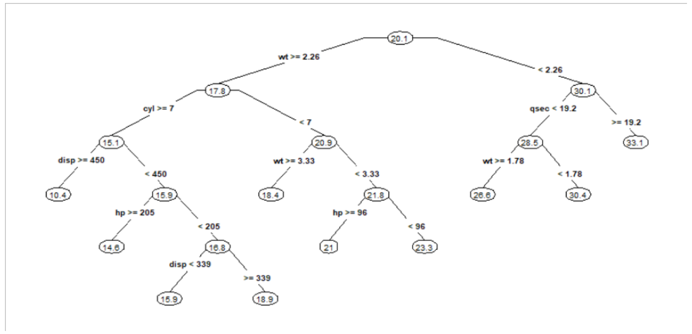
Introduction

- **What is Hyperparamter?**

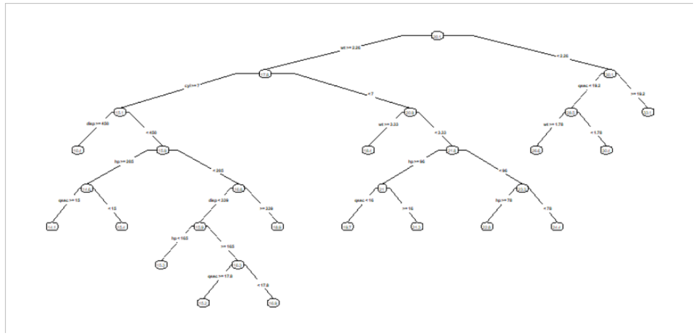
Hyperparameters is parameters which control the learning process of a model.

Ex) These Decision Trees are learned differently by setting the Hyperparameter (Cp, Complexity Upper bound)

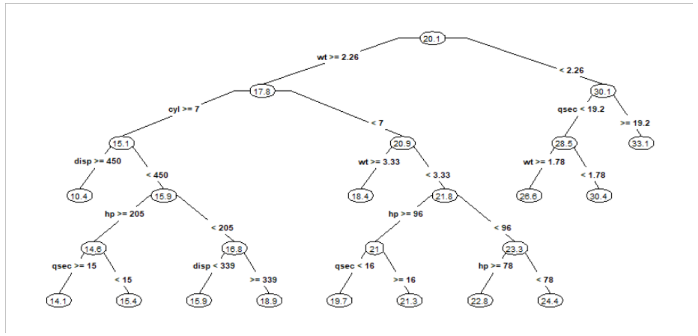
Even if the same input is used, different outputs are produced according to different hyperparameters.



Cp = 0.00095



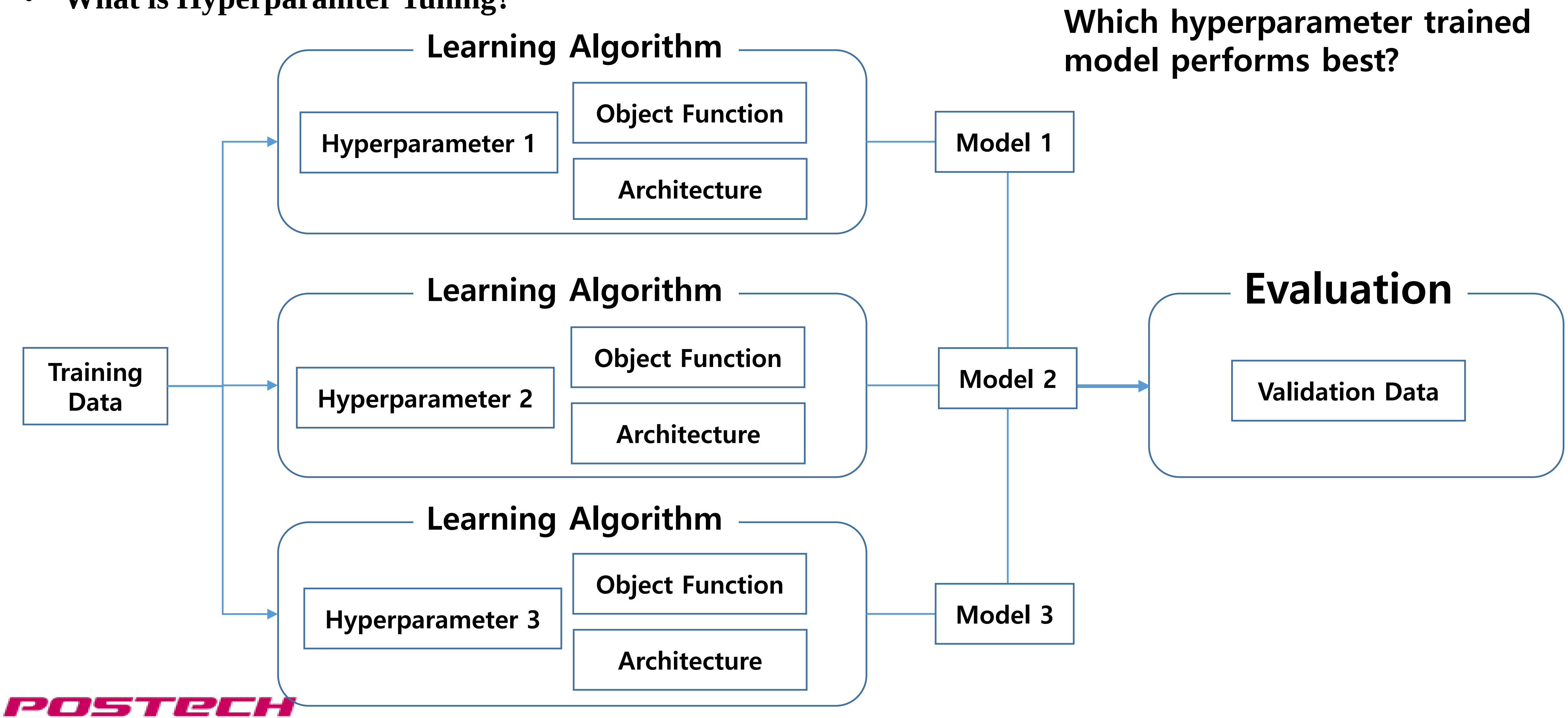
Cp = 0.001



Cp = 0.0015

Hyperparameter Tuning

- What is Hyperparamter Tuning?



Hyperparameter Tuning

There are two simple way for Hyperparameter Optimization.

1. Parallel Search : High computational cost
2. Sequential Design Strategy : High time complexity

Higher Score is better

	Worker 1	Worker 2	Worker 3	Worker 4	Worker 5
1 Trial	0.9	0.8	0.7	0.6	0.5
Score	10	20	60	30	10

Parallel Search (Drop out rate)

열1	Worker 1	Score
1 Trial	0.9	10
2 Trial	0.5	10
3 Trial	0.6	30
4 Trial	0.7	60

Sequential optimization (Drop out rate)

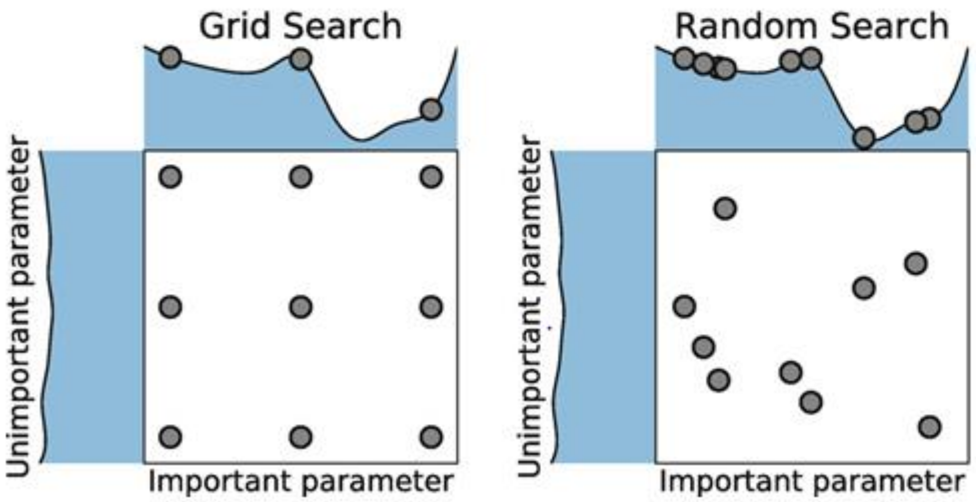
POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Hyperparameter Tuning

- **Parallel Search**

Grid Search is a method to find optimum values of hyperparameters by dividing hyperparameters into grid and calculating all combinations.

Random Search is a process of randomly selecting combination within the range of a defined Hyperparameter..



Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13, 281–305 (2012)

Hyperparameter Tuning

- **Parallel Search**

Random Search

Worker 1	**Trial 1** **Dropout : 0.84**	**Worker 1**	**Trial 2** **Dropout : 0.94**
Worker 2	**Trial 1** **Dropout : 0.96**	**Worker 2**	**Trial 2** **Dropout : 0.66**
Worker 3	**Trial 1** **Dropout : 0.72**	**Worker 3**	**Trial 2** **Dropout : 0.51**
		Worker 4	**Trial 2** **Dropout : random**

In parallel search, it is easy to add machine.

Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13, 281–305 (2012)

Hyperparameter Tuning

- **Sequential Design Strategy**

Determines the best choice from prior data.

a. Interpolation

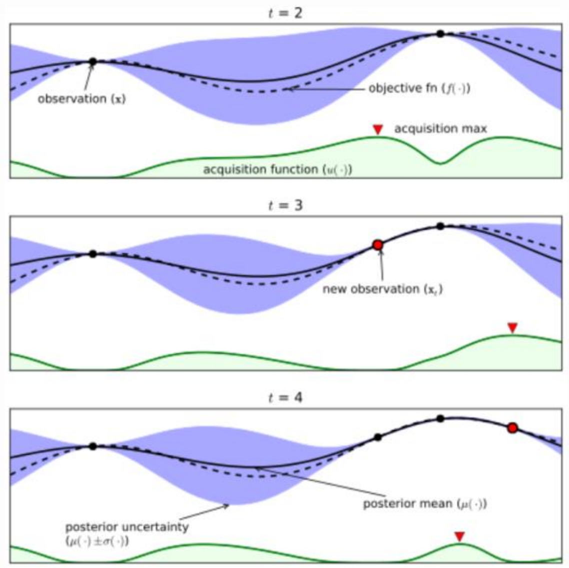
b. Bayesian Optimizaiton

1. Gaussian Process

2. TPE (Tree-structured Parzen Estimator Approach)

(Often used as a method of global surrogate modeling.)

(The amount of computation required in higher dimensions increases exponentially.)



Gang Lei, K. R. Shao, Youguang Guo, et al, (2008), "Sequential Optimization Method for the Design of Electromagnetic Device". IEEE

Hyperparameter Tuning

- **Population Based Trainning (PBT)**

Notation

Parameters : $\theta \in \Theta$

Hyperparameter : $h \in \mathcal{H}$, $\mathrm{H} = (h_1, h_2, h_3, \ldots) = (h_t)_{t=1}^{T} \in \mathcal{H}^T$, compact, convex set

Loss Function : $Loss$ (fixed)

Independent Variable : X, Dependent Variable : Y

Data : X_{train}, Y_{train}, X_{valid}, Y_{valid}, X_{test}, Y_{test}

Hyperparameter Tuning

- **Hyperparameter Optimization**

Random/Grid Search → Practical Bayesian optimization of machine learning algorithms(2012) → Successive Harving(2016) → Hyperband(2016) → PBT(2017) → Hyperband + Bayesian Optimization(Gaussian Process)(2018)→ Hyperband + Bayesian Optimization(TPE)(2018) → PBT + Bayesian Optimiation(Gaussian Process)(2020)

1. J. Sneok, H. Larochelle, R. P. Adams, (2012), “Practical Bayesian optimization of machine learning algorithms,” NIPS
2. K. Jamieson, A. Talwalkar, (2016), “Non-stochastic best arm identification and hyperparameter optimization,” AISTATS
3. Lisha Li, Kevin Jamieson, Giulia DeSalvo, et al., (2016), “Hyperband: A novel bandit-based approach to hyperparameter optimization,”, JMLR
4. Max Jaderberg, Valentin Dalibard, Simon Osindero, et al.(2017), Population Based Training of Neural Networks, NIPS
5. Klein, Stefan Falkner, Simon Bartels et al., (2018), “Fast Bayesian optimization of machine learning hyperparameters on large datasets,” , AISTATS
6. S. Falkner, A Klein, F. Hutter, (2018) “BOHB: robust and efficient hyperparameter optimization at scale,” ICML
7. Jack Parker-Holder, Vu Nguyen, Stephen Roberts (2020), Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, ICML

Hyperparameter Tuning

- **Population Based Trainning (PBT)**

Disadvantages of previous work

```
Search: Running Trial #1

Hyperparameter     |Value              |Best Value So Far
rnn_block_1/lay...|lstm               |?
rnn_block_1/num...|2                  |?
regression_head...|0.25               |?
optimizer         |adam               |?
learning_rate     |0.001              |?

Epoch 1/1000
281/281 [==============================] - 1s 5ms/step - loss: 926.1419 - mean_squ
ared_error: 926.1419 - val_loss: 539.9234 - val_mean_squared_error: 539.9234
```

```
Search: Running Trial #2

Hyperparameter     |Value              |Best Value So Far
rnn_block_1/lay...|lstm               |lstm
rnn_block_1/num...|2                  |2
regression_head...|0.25               |0.25
optimizer         |adam_weight_decay  |adam
learning_rate     |0.001              |0.001

Epoch 1/1000
281/281 [==============================] - 1s 5ms/step - loss: 990.5831 - mean_squ
ared_error: 990.5831 - val_loss: 635.9512 - val_mean_squared_error: 635.9512
```

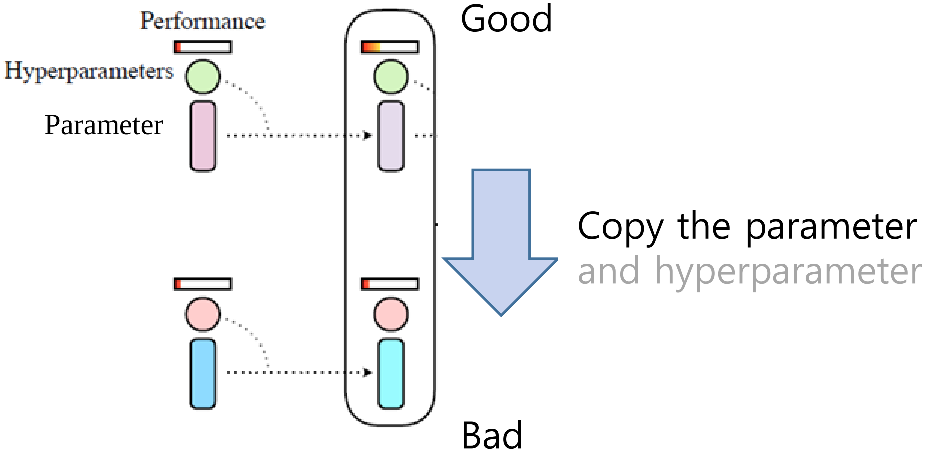
Does not reuse learned parameter when hyperparameters updated.

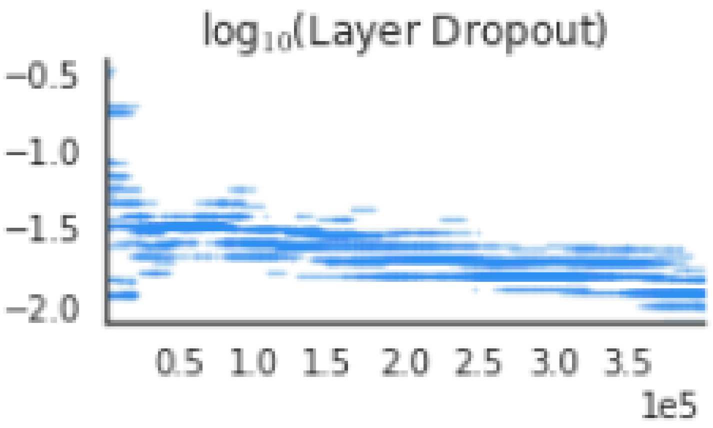
The Hyperparameter configuration is fixed during the training time

Hyperparameter Tuning

- **Population Based Trainning (PBT)**

What is PBT?





Copying the parameters of the model with good performance during training.

Change the Hyperparameter configuration during training

Max Jaderberg, Valentin Dalibard, Simon Osindero, et al.(2017), Population Based Training of Neural Networks, NIPS

Hyperparameter Tuning

- **Population Based Trainning (PBT)**

1. **Parameter update with hyperparameter.**

   step is Learning Process which updates parameters, θ_t with Hyperparameter, h_t.

   We change h_t every t_{ready} steps.

   $\theta_t = step(\theta_{t-1} \mid h_t)$

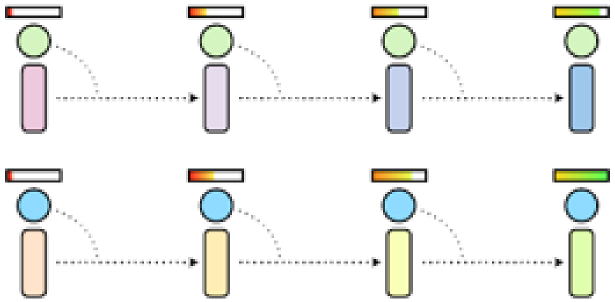
2. **Evaluation with Validation data**

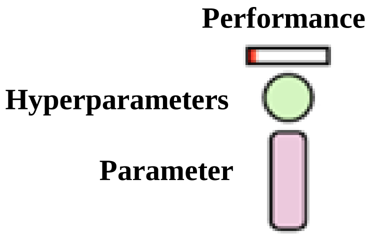
   eval is a function of trainable parameters.

   $eval(\theta) = Loss(f_\theta(X_{valid}), Y_{valid})$

Hyperparameter Tuning

- **Population Based Trainning (PBT)**





P = 2
T = 3

1. Set the size of population, P (Number of Worker).

2. Initialize each worker with different θ and h. $(\theta_i, h_i)_{i=1}^{P}$

3. Set the number of times to update hyperparameters, T

4. for t in T * t_{ready} **do**

$\theta_i \leftarrow step(\theta_i | h_i)$ in i=1,..,P

$p_i \leftarrow eval(\theta_i)$ in i=1,..,P

Denote $(\theta_i)_{i=1}^{P}$, $(h_i)_{i=1}^{P}$, $(p_i)_{i=1}^{P}$ as Θ, H, π

if t % t_{ready} **then**

H, Θ = exploit(H, Θ , π)

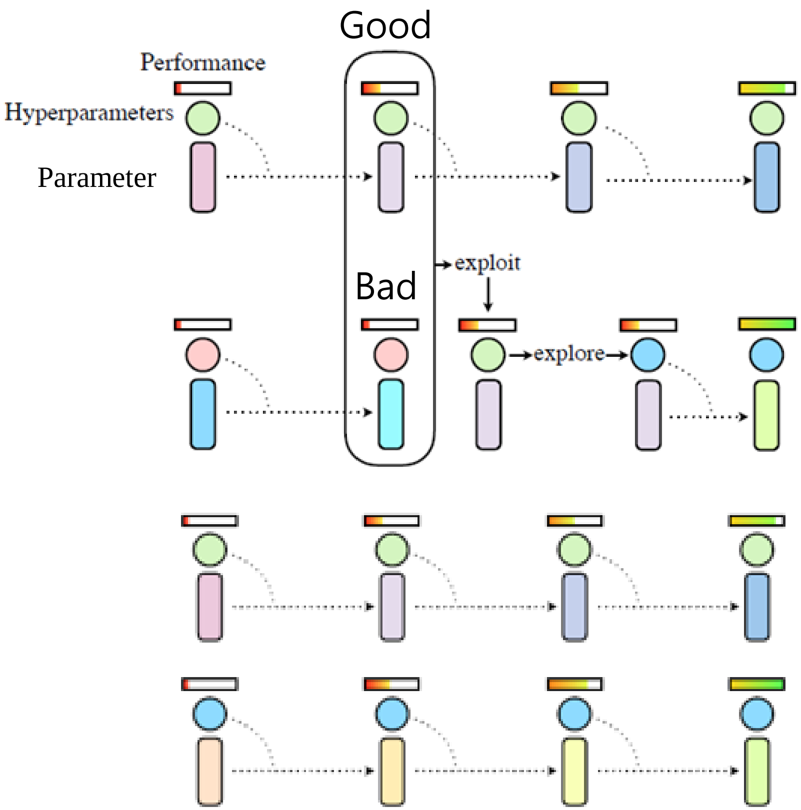
if exploit funtion is working **then**

H = explore(H, Θ , π)

Max Jaderberg, Valentin Dalibard, Simon Osindero, et al.(2017), Population Based Training of Neural Networks, NIPS

Hyperparameter Tuning

- **Population Based Trainning (PBT)**



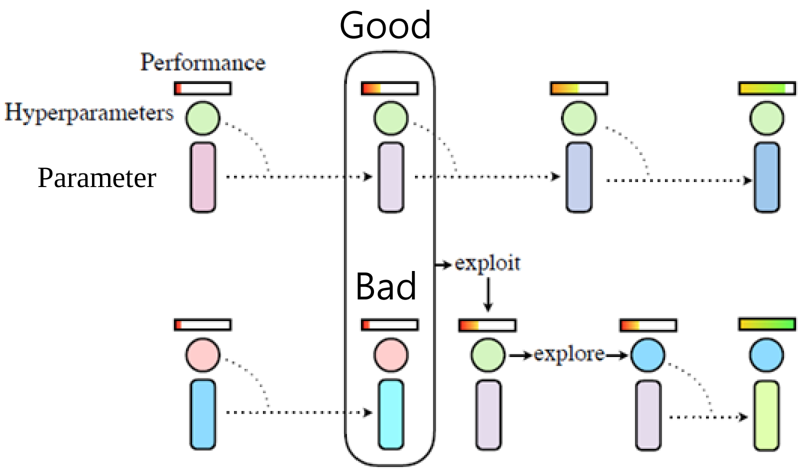
Exploit : If my worker's performance is bad, selects another member of the population to copy the parameters and hyperparameters.

Explore : Pertubate hyperparameters of exploited workers.

Max Jaderberg, Valentin Dalibard, Simon Osindero, et al.(2017), Population Based Training of Neural Networks, NIPS

Hyperparameter Tuning

- **Population Based Trainning (PBT)**



Exploit : If my worker's performance is bad, selects another member of the population to copy the parameters and hyperparameters.

Explore : Pertubate hyperparameters of exploited workers.

steps		Worker 1	Worker 2	Worker 3	Worker 4
1 * t_{ready}	Drop out	0.9	0.8	0.7	0.6
	Performance	10	30	60	40
	Weight	w1	w2	w3	w4

Exploit

steps		Worker 1	Worker 2	Worker 3	Worker 4
1 * t_{ready}	Drop out	0.7	0.8	0.7	0.6
	Performance	*	20	60	30
	Weight	w3	w2	w3	w4

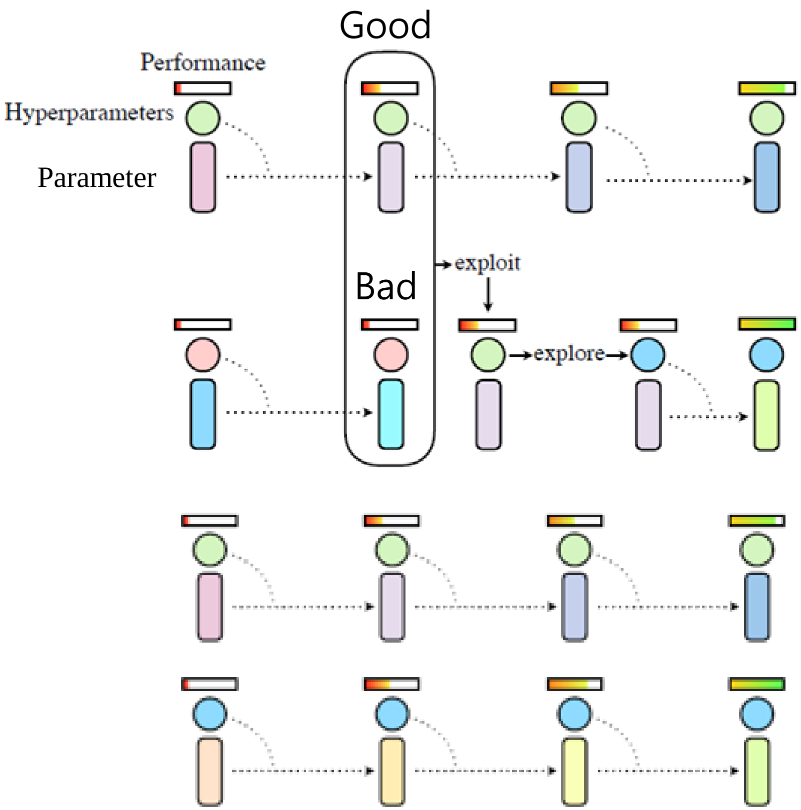
Explore

steps		Worker 1	Worker 2	Worker 3	Worker 4
1 * t_{ready}	Drop out	0.75	0.8	0.7	0.6
	Performance	*	20	60	30
	Weight	w3	w2	w3	w4

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Hyperparameter Tuning

- **Population Based Trainning (PBT)**



Exploit : If my worker's performance is bad, selects another member of the population to copy the parameters and hyperparameters.

1. **Binary tournament** : If the performance is worse than the randomly selected worker, copy the θ and h.
2. **T-test Selection :** Sample the last 10 performance with randomly selected worker., do the t-test
3. **Truncation selection** : If the performance is the bottom 20% of all workers, randomly choose one of the top 20% workers and copy the θ and h.

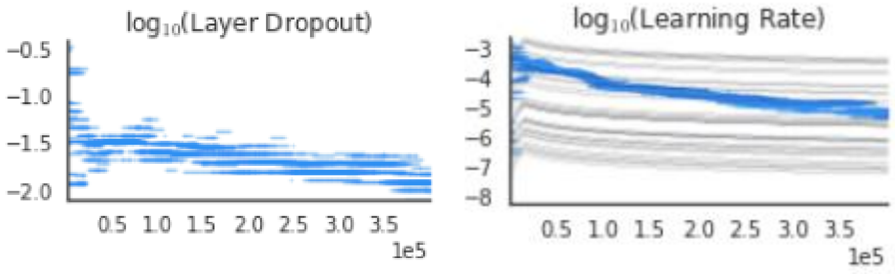
Explore : Pertubate hyperparameters of exploited workers.

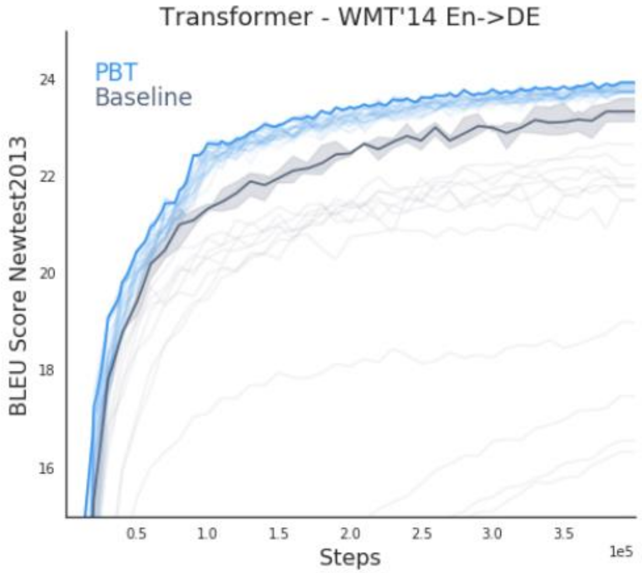
1. **Perturb** : Each hyperparameter independently is randomly perturbed by a factor of 1.2 or 0.8.
2. **Resample** : Where each hyperparameter is resampled from the original prior distribution defined with some probability.(log-uniform)

Max Jaderberg, Valentin Dalibard, Simon Osindero, et al.(2017), Population Based Training of Neural Networks, NIPS

Hyperparameter Tuning

- **Population Based Trainning (PBT)**
  - PBT for Machine Translation
  - Model
    - WMT 2014 English-to-German, Transformer networks (Vaswani et al., 2017),
  - Hyperparameter
    - learning rate, attention dropout, layer dropout, and ReLU dropout rates.
  - Step
    - gradient descent with Adam (Kingma & Ba, 2015).
    - Exploit every 2×10^3 steps, total 400×10^3 steps
  - Baseline
    - The model found by random search.

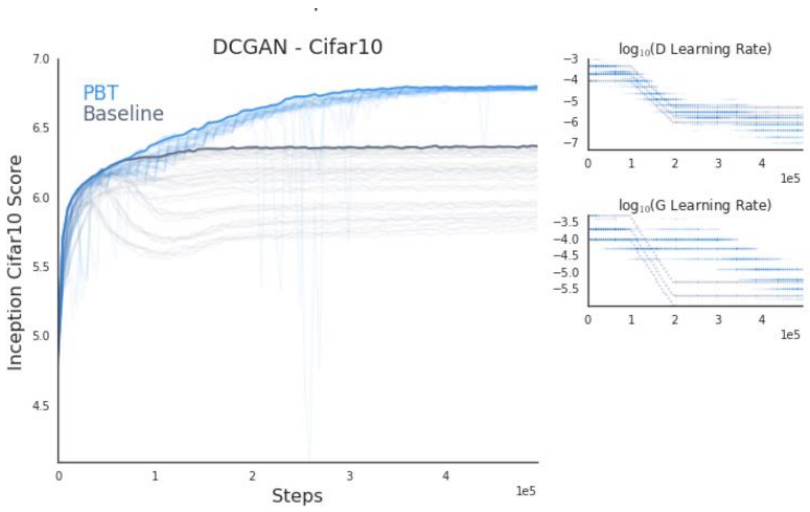


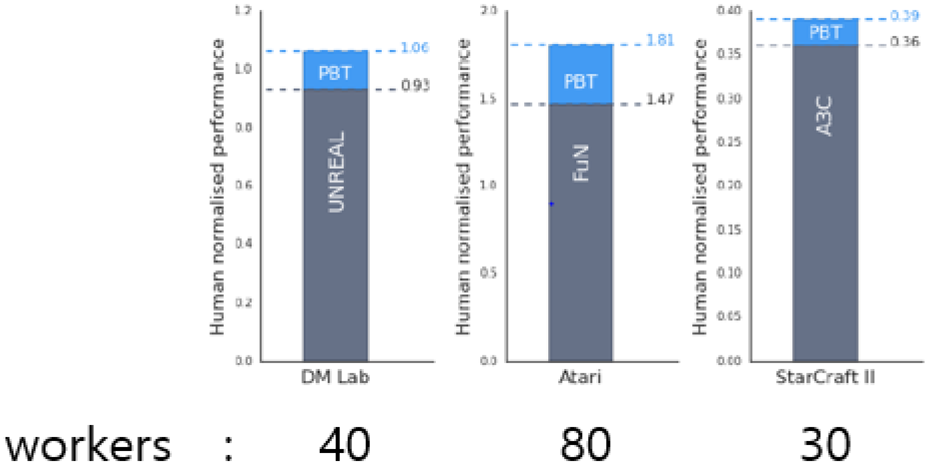


Ashish Vaswani, Noam Shazeer, Niki Parmar et al.(2017) Attention is all you need. NIPS

Hyperparameter Tuning

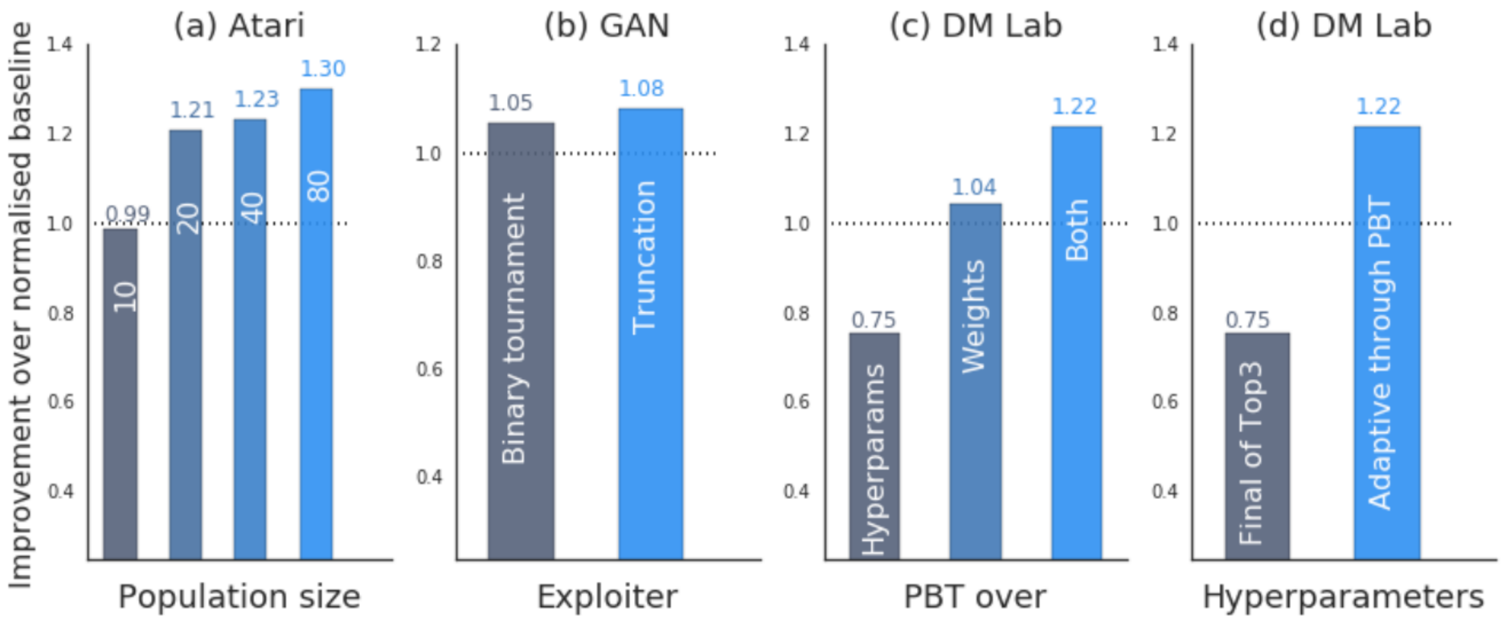
- **Population Based Trainning (PBT)**
  - PBT for Reinforce Learning
  - 3 tasks and models
    - DeepMind Lab, UNREAL (Jaderberg et al., 2016)
    - Atari games, Feudal Networks (Vezhnevets et al., 2017)
    - StarCraft 2, A3C baseline agents (Vinyals et al., 2017)
  - Hyperparameter
    - learning rate, entropy cost, unroll length for UNREAL on DeepMind Lab, intrinsic reward cost for FuN on Atari
  - Step
    - Step Each iteration does a step of gradient descent with RMSProp (Tieleman & Hinton, 2012)
    - Exploit every 10^6 steps, total $2.5 * 10^8$ steps
  - Baseline
    - The model found by random search.





Hyperparameter Tuning

- **Population Based Trainning (PBT)**
  - A. Population Size
  - B. Strategy of Exploiter
  - C. Transfer Learning
  - D. Importance of the adaptation



equivalent random search baseline.

Hyperparameter Tuning

- **Population Based Bandits (PB2)**
  - Disadvantages of PBT:
  - PBT relies on heuristics to explore the hyperparameter space
    - Lacks theoretical guarantees.
    - Requires vast computational resources → Small populations will often collapse to a suboptimal mode,
  - What is PB2?
  - New PBT using Bayesian Optimization technique
  - → Good performance is obtained with relatively few workers, and there is a theoretical guarantee for performance.
  - → They use time-variant and batch Bayesian Optimization .

Jack Parker-Holder, Vu Nguyen, Stephen Roberts (2020)
Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, ICML

Hyperparameter Tuning

- **Population Based Bandits (PB2)**

1. Set the size of population, P (Number of Worker).

2. Initialize each worker with different θ and h. $(\theta_i, h_i)_{i=1}^{P}$

3. Set the number of times to update hyperparameters, T

4. For t in T do

   $\theta_i \leftarrow \text{step}(\theta_i | h_i)$ in i=1,..,P

   $p_i \leftarrow \text{eval}(\theta_i)$ in i=1,..,P

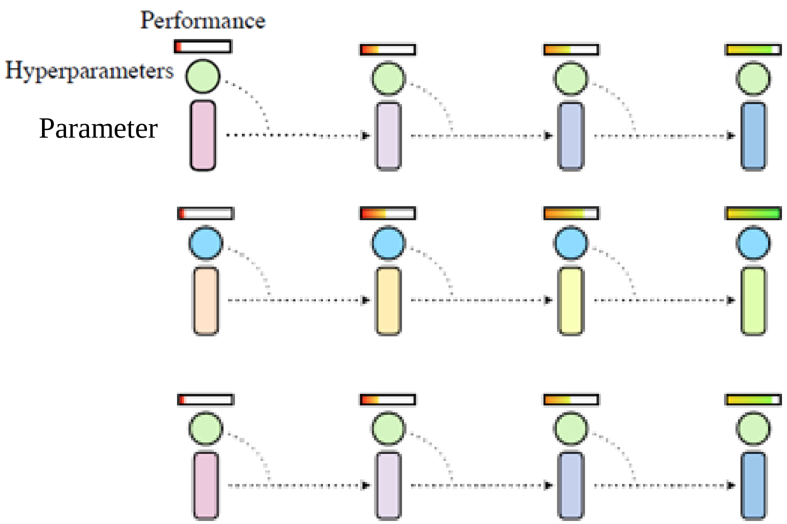
   Denote $(\theta_i)_{i=1}^{P}$, $(h_i)_{i=1}^{P}$, $(p_i)_{i=1}^{P}$ as Θ, H, π

   H, Θ = exploit(H, Θ , π)

   If exploit funtion is working.

   $\boxed{H = \text{explore}(H, \Theta, \pi)}$

Bayesian Optimization



Truncation selection : If the performance is the bottom 20% of all workers, randomly choose one of the top 20% workers and copy the θ and h

Hyperparameter Tuning

- **Population Based Bandits (PB2)**
  - What is Bandits? --> Multi-armed Bandits Problem

  - There are K possible actions (arms), and the algorithm gets reward by selecting one action.
  - Reward follows a stationary probability distribution.
  - The goal is to maximize the expected total reward for a limited time.

Num	1	2	3	4	5
Reward	?	?	3$	4$	?
# of trial	0	0	1	3	0

Explore

Num	1	2	3	4	5
Reward	?	?	3$	4$	5$
# of trial	0	0	1	3	1

Exploit

Num	1	2	3	4	5
Reward	?	?	3$	4$	?
# of trial	0	0	1	4	0

Aleksandrs Silvkins, (2019), Introduction to Multi-Armed Bandits, Microsoft Research NYC

Hyperparameter Tuning

- **Population Based Bandits (PB2)**
  - **Notation**
  - $f_T(h_T) = \text{eval}(\theta_T) - \text{eval}(\theta_{T-1})$
  - $h_t{}^* = \underset{h \in \mathcal{H}}{\text{argmax}}\ f_t(h_t)$, best choice at each timestep.
  - $r_t = f_t(h_t *) - f_t(h_t)$, regret
  - $R_T = \sum_{t=1}^{T} r_t$, curmulative regret

Step(T)	1	2	3	4	5
Performance	10	50	60	65	68
$f_T(h_T)$		40	10	5	3

- Parameters : $\theta \in \Theta$
- Hyperparameter : $h \in \mathcal{H}$
- Evaluation: eval(θ)
- Parameter update :

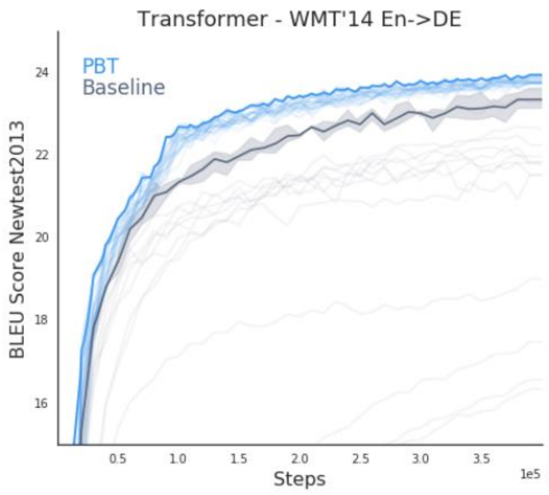
  $\theta \leftarrow \text{step}(\theta|h)$

Jack Parker-Holder, Vu Nguyen, Stephen Roberts (2020)
Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, ICML

Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - We gonna use Bayesian Optimization for estimating $\mathrm{f_T}(\mathrm{h_T})$, $\mathrm{f_T}(\mathrm{h_T}) = \mathrm{eval}(\theta_\mathrm{T}) - \mathrm{eval}(\theta_{\mathrm{T}-1})$.
  - The GP posterior belief at new point $\mathrm{f_T}(\mathrm{h_T}')$ follows a Gaussian Distribution with mean $\mu_t(\mathrm{h_T}')$ and variance $\sigma^2{}_t(\mathrm{h_T}')$.
  - $\mu_t(\mathrm{h_T}') = k_t(\mathrm{h_T}')^\mathrm{T}(K_t+\sigma^2\mathrm{I})^{-1}y_t$
  - $\sigma^2{}_t(\mathrm{h_T}') = \mathrm{k}(\mathrm{h_T}', \mathrm{h_T}') - k_t(\mathrm{h_T}')^\mathrm{T}\,(K_t+\sigma^2\mathrm{I})^{-1}k_t(\mathrm{h_T}')$
  - where $y_T = ((\mathrm{f}_t(\mathrm{h}_t))_{t=1}^T)^\mathrm{T}$, $K_t = (k\ (\mathrm{h}_i, \mathrm{h}_j))_{i,j=1}^T$ and $k_T = (k\ (\mathrm{h}_i, \mathrm{h}_T'))_{i=1}^T$
  - $\mathrm{h_T}'$, which maximize acquisition function becomes $\mathrm{h_{T+1}}$.



Jack Parker-Holder, Vu Nguyen, Stephen Roberts (2020)
Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, ICML

Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - For theoretical guarantee for performance, we use GP-UCB

  - **GP-UCB**
  - $h_{T+1} = \underset{h \in \mathcal{H}^T}{\operatorname{argmax}} \; \mu_T(h_T) + \sigma_T(h_T) * \sqrt{\beta_T}$
  - GP-UCB holds sum of cumulative regret's upper bound with some probability.

  - Lemma 1
  - $\max \text{eval}(\theta_T) = \max \sum_{t=1}^{T} f_t(h_t) = \min \sum_{t=1}^{T} r_t = \min R_T$

    Final reward

    Sum of cumulative regret

Jack Parker-Holder, Vu Nguyen, Stephen Roberts (2020) Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, ICML

Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - Suppose that the kernel satisfies the continuously differentiable and lipschitz asumption in a compact and convex domain.
  - The sum of the regrets of the PB2 algorithm has the following upper bound with a probability of at least 1-δ after T time. (δ is a variable related to β_T)
  - The higher δ, the less exploration and the smaller the bound.

$$R_T = \sum_{t=1}^{T} f_t(\mathbf{x}_t^*) - f_t(\mathbf{x}_t) \leq \sqrt{C_1 T \beta_T \left(\frac{T}{\tilde{N}B} + 1\right)\left(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^{\frac{5}{2}} \omega\right)} + 2$$

Sum of cumulative regret

Bound

$$\beta_T = 2\log\frac{\pi^2 T^2}{2\delta} + 2d\log rdbT^2\sqrt{\log\frac{da\pi^2 T^2}{2\delta}}$$

Jack Parker-Holder, Vu Nguyen, Stephen Roberts (2020)
Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, ICML

Hyperparameter Tuning

- **Population Based Bandits (PB2)**

- Suppose that the kernel satisfies the continuously differentiable and lipschitz asumption in a compact and convex domain.
- The sum of the regrets of the PB2 algorithm has the following upper bound with a probability of at least 1-δ after T time. (δ is a variable related to β_T)
- The higher δ, the less exploration and the smaller the bound.

More exploration → Small bound with small probability

Good upper bound has small probability.

$$R_T = \sum_{t=1}^{T} f_t(\mathbf{x}_t^*) - f_t(\mathbf{x}_t) \leq \sqrt{C_1 T \beta_T \left(\frac{T}{\tilde{N}B} + 1\right)\left(\gamma_{\tilde{N}B} + \left[\tilde{N}B\right]^{\frac{5}{2}} \omega\right)} + 2$$

Sum of cumulative regret

Bound

$$\beta_T = 2 \log \frac{\pi^2 T^2}{2\delta} + 2d \log rdbT^2 \sqrt{\log \frac{da\pi^2 T^2}{2\delta}}$$

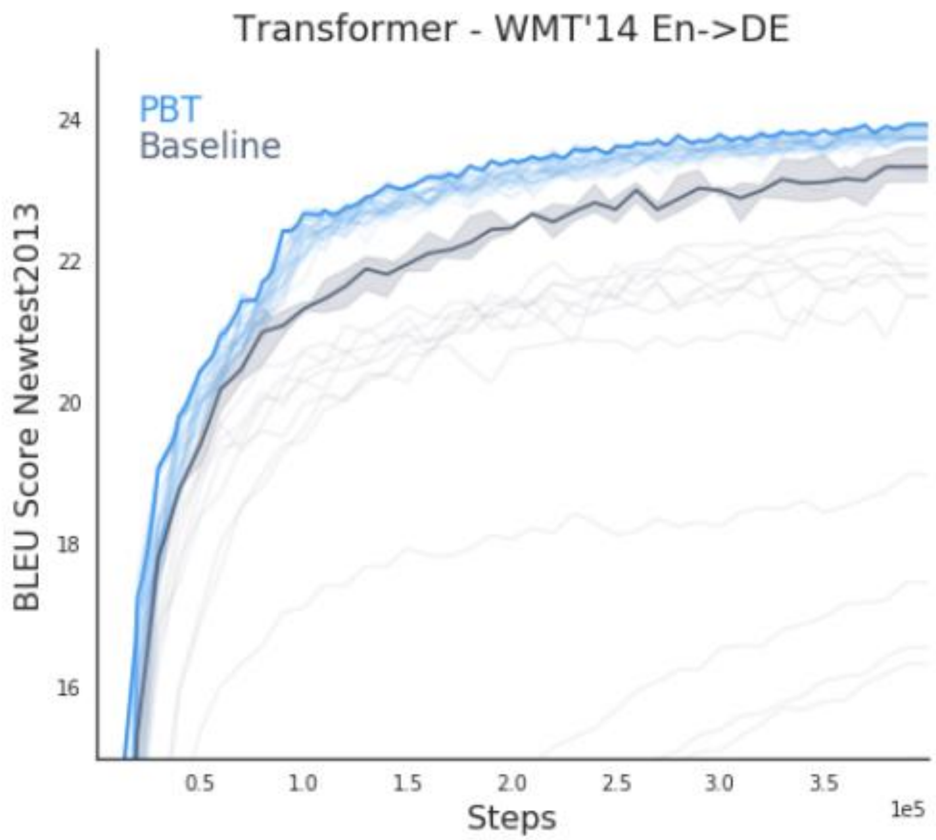
Jack Parker-Holder, Vu Nguyen, Stephen Roberts (2020)
Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, ICML

Hyperparameter Tuning

- **Population Based Bandits (PB2)**
  - **Notation**
  - $f_T(h_T) = \text{eval}(\theta_T) - \text{eval}(\theta_{T-1})$

Step	1	2	3	4	5
Performance	10	50	60	65	68
$f_T(h_T)$		40	10	5	3

  - Recorded Data ($f_T(h_T)$, h_T, T) →
  - We model $f_T(h_T)$) using a time-varying Gaussian Process.



Jack Parker-Holder, Vu Nguyen, Stephen Roberts (2020) Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, ICML

Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - How to cast the problem of optimizing the problem as time-varying optimization?

  - $f_1(h) = g_1(h)$
  - $f_{T+1}(h) = \sqrt{1-w}\, f_T(h) + \sqrt{w}\, g_T(h)$
  - Where g_1, g_2, … are independent random functions with g_t ~ GP(0, k)

  - $w = 0$ → time-unvarying GP-UCB
  - $w = 1$ → f are independent between time steps. (The covariance matrix is diagonal.)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al.(2015) Deep Residual Learning for Image Recognition. CVPR

Ilija Bogunovic, Jonathan Scarlett, and Volkan Cevher.(2016) Time-Varying Gaussian Process Bandit Optimization. AISTATS

Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - $f_{T+1}(h) = \sqrt{1-w}\, f_T(h) + \sqrt{w}\, g_T(h)$
  - Where g_1, g_2, … are independent random functions with g_t ~ GP(0, k)

  - $K_t = (k\,(h_i, h_j))_{i,j=1}^{T}$ and $k_T = (k\,(h_i, h_T{}'))_{i=1}^{T}$ is changed to $K_t{}'$, $k_t{}'$
  - $K_t{}' = K_t \circ K_t^{time}$ where $K_t^{time} = ((1-w)^{|i-j|/2})_{i,j=1}^{T}$
  - $k_t{}' = k_t \circ k_t^{time}$ where $k_t^{time} = ((1-w)^{|T+1-i|/2})_{i=1}^{T}$
  - ◦ refers to the Hadmard product (elementwise product).

Ilija Bogunovic, Jonathan Scarlett, and Volkan Cevher.(2016) Time-Varying Gaussian Process Bandit Optimization. AISTATS

Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - For theoretical guarantee for performance, we use GP-UCB

  - **GP-UCB**
  - $h_{T+1} = \underset{h \in \mathcal{H}^T}{\mathrm{argmax}} \ \mu_T(h_T) + \sigma_T(h_T) * \sqrt{\beta_T}$
  - GP-UCB holds sum of cumulative regret's upper bound with some probability.

  - Lemma 1
  - $\max \mathrm{eval}(\theta_T) = \max \sum_{t=1}^{T} f_t(h_t) = \min \sum_{t=1}^{T} r_t = \min R_T$

Final reward

Sum of cumulative regret

Jack Parker-Holder, Vu Nguyen, Stephen Roberts (2020)
Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, ICML

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

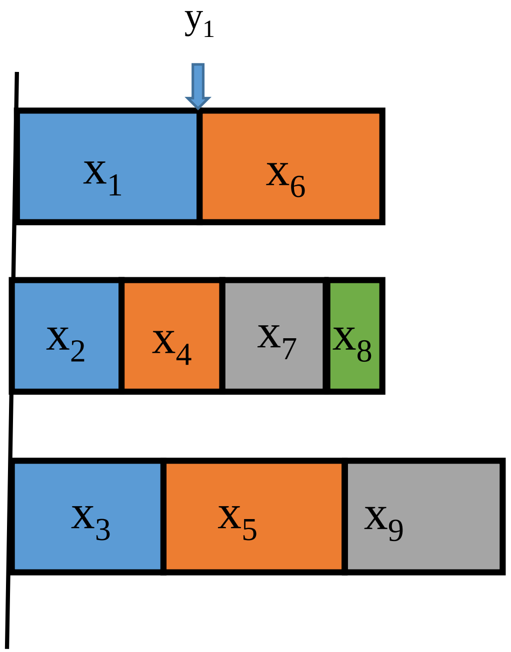
Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - Batch blackbox optimization problem

  - Bayesian Optimization use posterior (x and y). But if we do batch optimization, we must select x_t without full knowledge of all $(xi,yi)_{i=1}^{t-1}$.

  - For example, if we want to find x_7 which maximize the acquisition function, we can use $x_1,\ldots x_6$ and $y_1,..,y_4$.



Thomas Desautels, Andreas Krause, and Joel W. Burdick.(2014) Parallelizing exploration-exploitation tradeoffs in Gaussian Process bandit optimization, JMLR

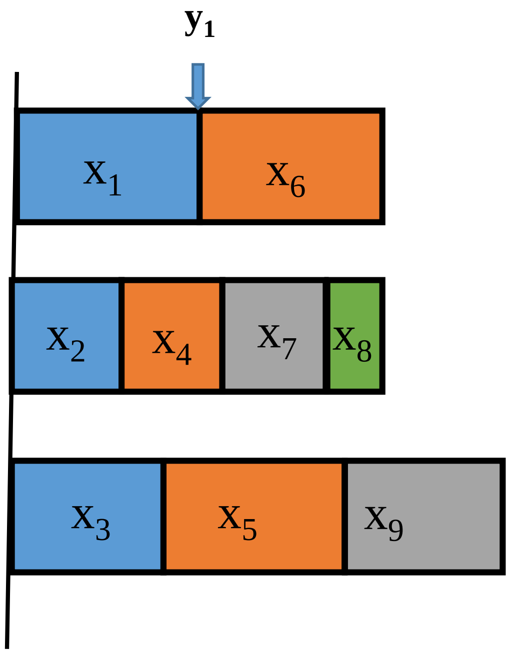
Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - Batch blackbox optimization problem

  - $h_{T+1} = \underset{h \in \mathcal{H}^T}{\operatorname{argmax}} \ \mu_T(h_T) + \sigma_T(h_T) * \sqrt{\beta_T}$

  - When we calculate $\sigma_T(h_T)$, we only use $x_1, \dots, x_T$, not y.
  - So $\mu_6(h_6)$ is calculated with $(xi,yi)_{i=1}^{4}$ and $\sigma_T(h_T)$ is with $(xi)_{i=1}^{6}$



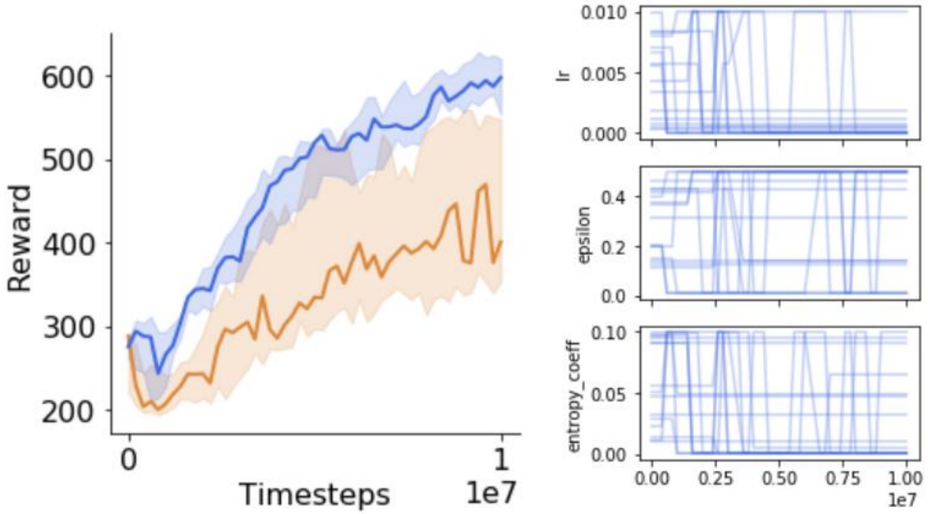
Thomas Desautels, Andreas Krause, and Joel W. Burdick.(2014) Parallelizing exploration-exploitation tradeoffs in Gaussian Process bandit optimization, JMLR

Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - Learning Algorithm
    - IMPALA(Importance parametered Actor-Learner Architecture) with PB2 with population 4
  - Model
    - Space Invaders. (Bellemare et al., 2012).
  - Hyperparameter

Parameter	Value
Epsilon	$\{0.01, 0.5\}$
Learning Rate	$\{10^{-3}, 10^{-5}\}$
Entropy Coeff	$\{0.001, 0.1\}$



  - Step
    - Exploit every 5×10^5 steps, total 1×10^7 steps
  - Baseline
    - IMPALA with PBT with population 24.

Lasse Espeholt, Hubert Soyer, Remi Munos. et al.(2018) IMPALA: Scalable distributed deep-RL with importance parametered actor-learner architectures. ICML.

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

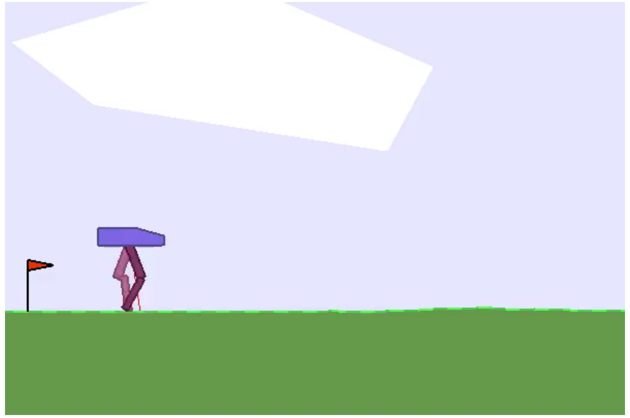
Hyperparameter Tuning

- **Population Based Bandits (PB2)**

  - Optimizing following hyperparameter: batch size, learning rate, GAE parameter and PPO clip parameter
  - PB2, PBT, RS, ASHA seek to optimize hyperparameter for Proximal Policy Optimization (PPO, Schulman et al. (2017)).

Table 1: Median best performing agent across 10 seeds. The best performing algorithms are bolded.

	B	RS	ASHA	PBT	PB2	vs. PBT
BipedalWalker	4	234	236	223	**276**	+24%
LunarLanderContinuous	4	161	213	159	**235**	+48%
Hopper	4	1638	1819	1492	**2346**	+57%
InvertedDoublePendulum	4	8094	7899	**8893**	8179	-8%
BipedalWalker	8	240	255	277	**291**	+5%
LunarLanderContinuous	8	175	231	247	**275**	+11%

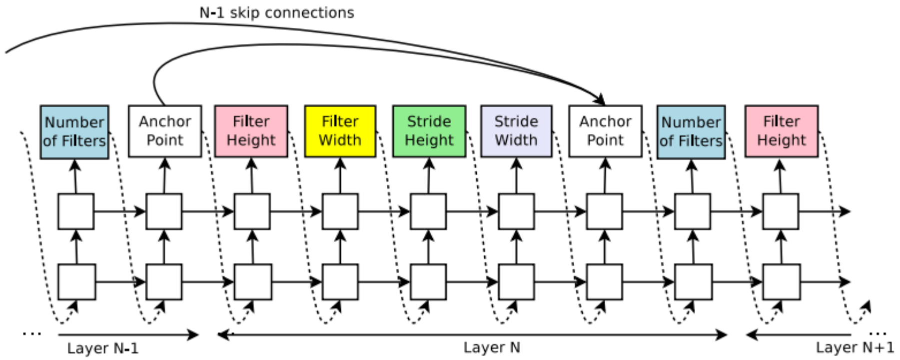


OpenAI Gym (Brockman et al., 2016),

BipedalWalker

Performance of AutoML

- **Neural Architecture Searching (NAS)**
  - A method of finding the optimal Deep Learning Architecture based on reinforcement learning.
  - Changing Hyperparameter:
  - filter height, filter width, stride height, stride width, and number of filters for one layer, skip connection
  - Create a Hyperparameter that composes a Deep Learning Model through RNN Model.

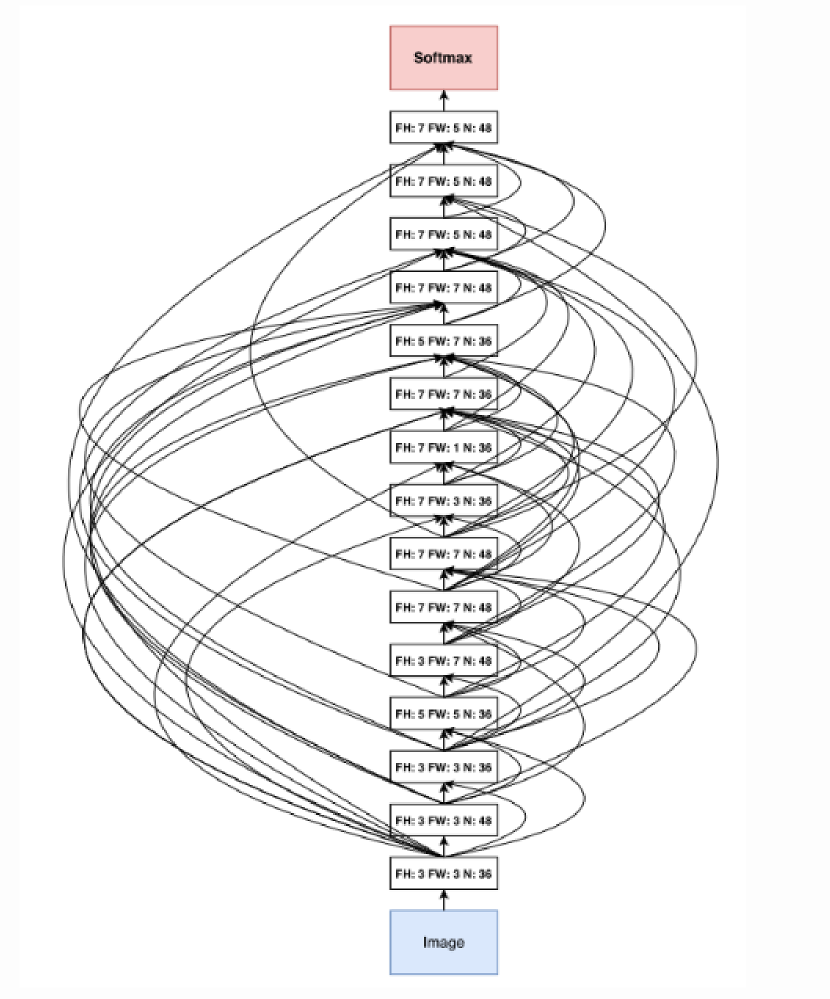


Barret Zoph, Quoc Le (2017)
Neural Architecture Search with Reinforcement Learning, ICLR

Performance of AutoML

- **Neural Architecture Searching (NAS)**

    - 800 GPU, 28 days (NVIDIA K40 GPU)
    - 2 x 32 size, 50000 images (CIFAR-10)
    - Performance : Similar to ResNet and similar performance to DenseNet.
    - Even though the image is not large, it takes a lot of time. It is difficult to apply to various image sets.

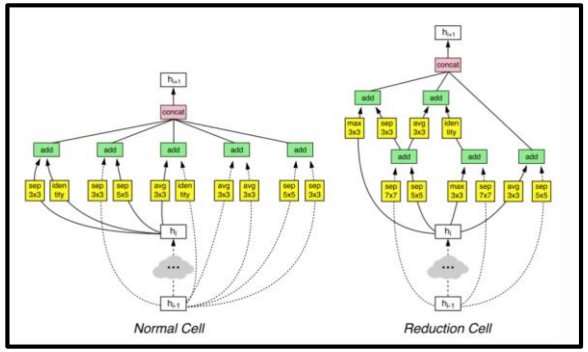


Barret Zoph, Quoc Le (2017)
Neural Architecture Search with Reinforcement Learning, ICLR

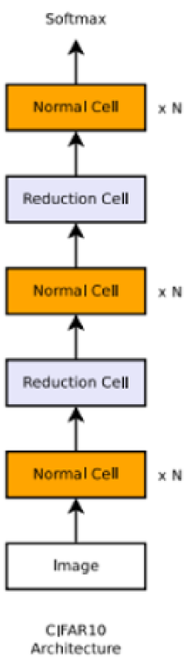
Performance of AutoML

- **Learning Transferable Architectures (NASNET)**

    - Through RNN, let's not control every part of Architecture in detail, but make it in a specific unit!



Cell



Architecture

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, etc (2018)
Learning Transferable Architectures for Scalable Image Recognition, CVPR

Performance of AutoML

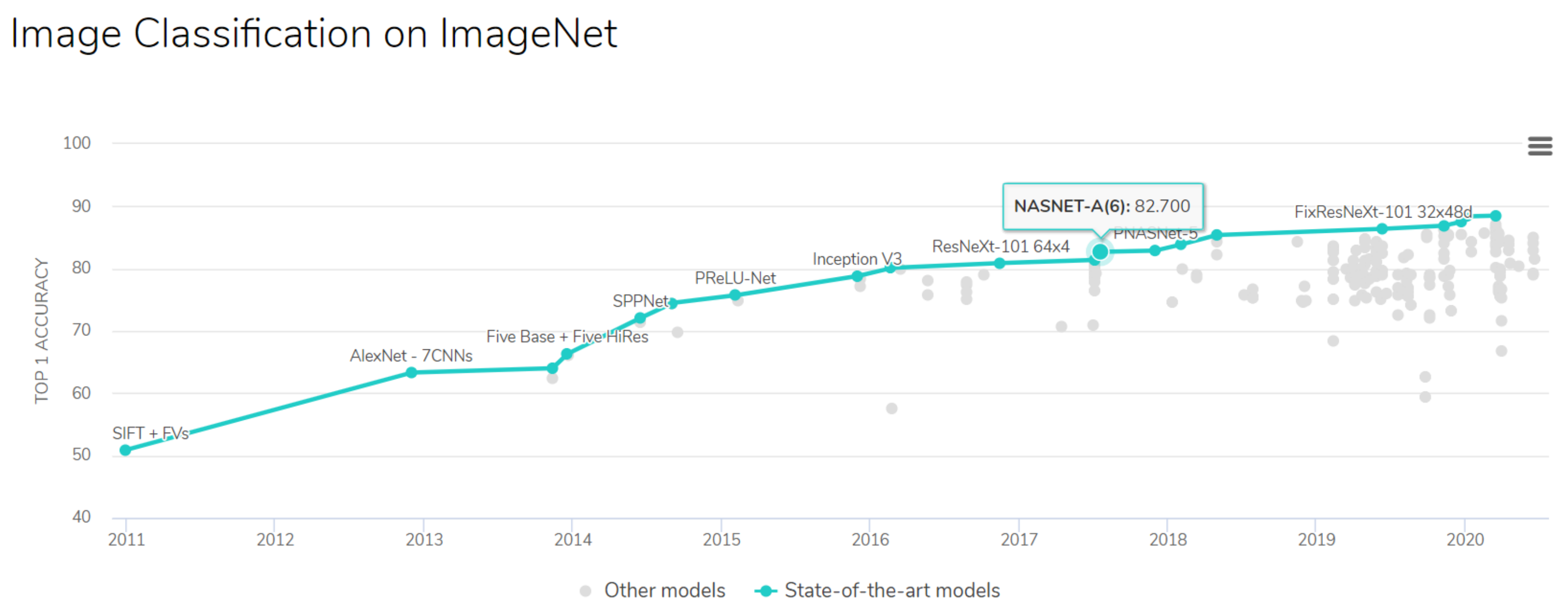
- **Learning Transferable Architectures (NASNET)**

  - CIFAR : 10 types, 50,000 images, sizes are different 32 x 32 size
  - ImageNet : 1000 types, 1,281,167 images, sizes are different 256 x 256 size

Performance of AutoML

- **Learning Transferable Architectures (NASNET)**

  - There was no case of presenting the results of applying AutoML to ImageNet.
  - Shows the same performance as SENet, which won the 2017 ImageNet challenge.
  - Besides this, it shows good performance for various data and purposes such as object detection.



1000 types, 1,281,167 images, sizes are different 256 x 256 size

Performance of AutoML

- **AutoGluon-Tabular**
  - Faster, Robust, Accurate compared to other AutoML platforms (TPOT, H2O, AutoWEKA, auto-sklearn, and GCP AutoML).
  - Participated in two famous Kaggle competitions and produced a better model than 99% of data scientists. (Uses 4 hours each)

	Framework	Wins	Losses	Failures	Champion	Rank	Performance	Time
OpenML	AutoGluon	-	-	**1**	**23**	**1.8438**	**0.8615**	201
	H2O AutoML	4	26	8	2	3.1250	0.7553	220
	TPOT	6	27	5	5	3.3750	0.7966	235
	GCP-Tables	5	20	14	4	3.7500	0.6664	**195**
	auto-sklearn	6	27	6	3	3.8125	0.6803	240
	Auto-WEKA	4	28	6	1	5.0938	0.1999	244
Kaggle	AutoGluon	-	-	**0**	**7**	**1.7143**	**0.7041**	**202**
	GCP-Tables	3	7	1	3	2.2857	0.6281	222
	H2O AutoML	1	7	3	0	3.4286	0.5129	227
	TPOT	1	9	1	0	3.7143	0.4711	380
	auto-sklearn	3	8	**0**	1	3.8571	0.4819	240
	Auto-WEKA	0	10	1	0	6.0000	0.2056	221

Competition	Task	Metric	Year	Teams	Rows	Colums
house-prices-advanced-regression-techniques	regression	RMSLE	2020	5100	1460	80
mercedes-benz-greener-manufacturing	regression	R^2	2017	3800	4209	377
santander-value-prediction-challenge	regression	RMSLE	2019	4500	4459	4992
allstate-claims-severity	regression	MAE	2017	3000	1.8E+5	131
bnp-paribas-cardif-claims-management	binary	log-loss	2016	2900	1.1E+5	132
santander-customer-transaction-prediction	binary	AUC	2019	8800	2.2E+5	201
santander-customer-satisfaction	binary	AUC	2016	5100	7.6E+4	370
porto-seguro-safe-driver-prediction	binary	Gini	2018	5200	6.0E+5	58
ieee-fraud-detection	binary	AUC	2019	6400	5.9E+5	432
walmart-recruiting-trip-type-classification	multi-class	log-loss	2016	1000	6.5E+5	7
otto-group-product-classification-challenge	multi-class	log-loss	2015	3500	6.2E+4	94

Nick Erickson, Jonas Mueller, Alexander Shirkov (2020). AutoGluon-Tabular: Robust and Accurate. AutoML for Structured Data,ICML

Performance of AutoML

- **AutoGluon-Tabular**
  - Neural networks
  - LightGBM boosted trees (Ke et al., 2017),
  - CatBoost boosted trees (Prokhorenkova et al., 2018)
  - scikit-learn implementations of: Random Forests
  - Extremely Randomized Trees,
  - kNearest Neighbors.
  - Use Stack Ensembling

Competition	Task	Metric	Year	Teams	Rows	Colums
house-prices-advanced-regression-techniques	regression	RMSLE	2020	5100	1460	80
mercedes-benz-greener-manufacturing	regression	R^2	2017	3800	4209	377
santander-value-prediction-challenge	regression	RMSLE	2019	4500	4459	4992
allstate-claims-severity	regression	MAE	2017	3000	1.8E+5	131
bnp-paribas-cardif-claims-management	binary	log-loss	2016	2900	1.1E+5	132
santander-customer-transaction-prediction	binary	AUC	2019	8800	2.2E+5	201
santander-customer-satisfaction	binary	AUC	2016	5100	7.6E+4	370
porto-seguro-safe-driver-prediction	binary	Gini	2018	5200	6.0E+5	58
ieee-fraud-detection	binary	AUC	2019	6400	5.9E+5	432
walmart-recruiting-trip-type-classification	multi-class	log-loss	2016	1000	6.5E+5	7
otto-group-product-classification-challenge	multi-class	log-loss	2015	3500	6.2E+4	94