Introduction to AutoML

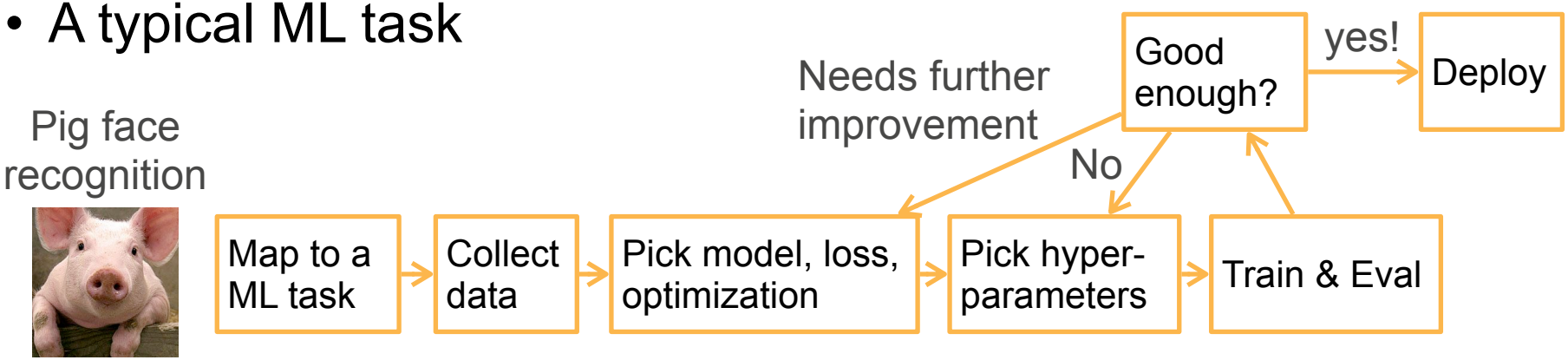
Mu Li (mli@amazon.com)

April 2020

© 2020, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

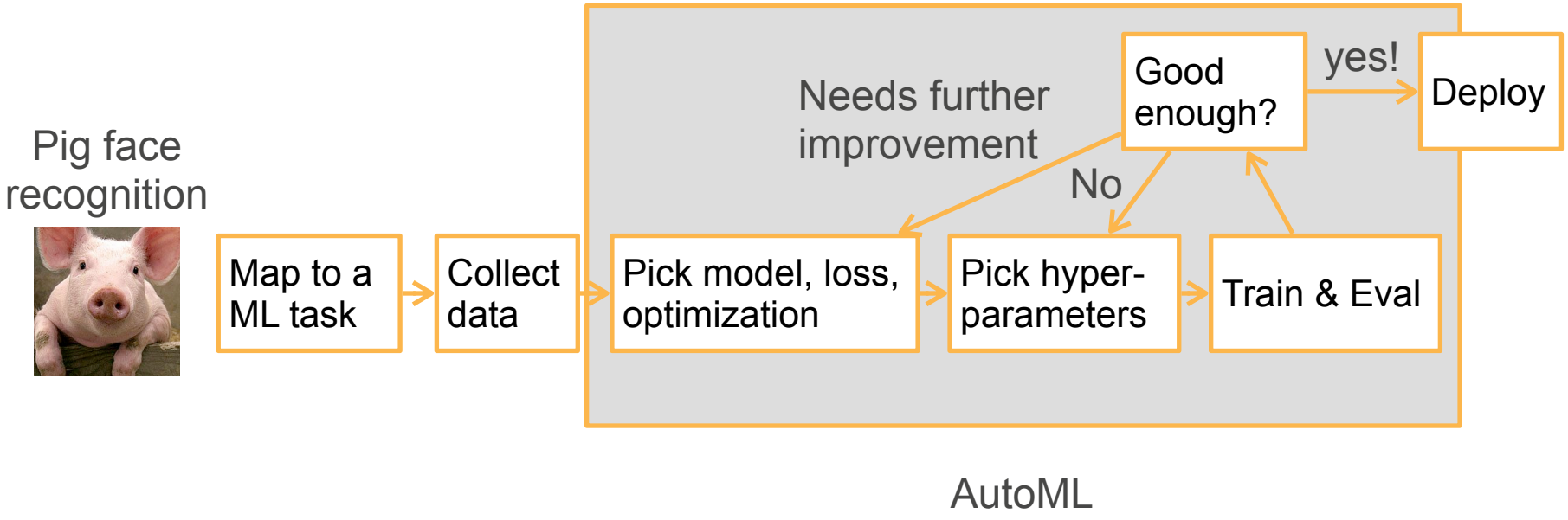
Machine Learning

- A typical ML task



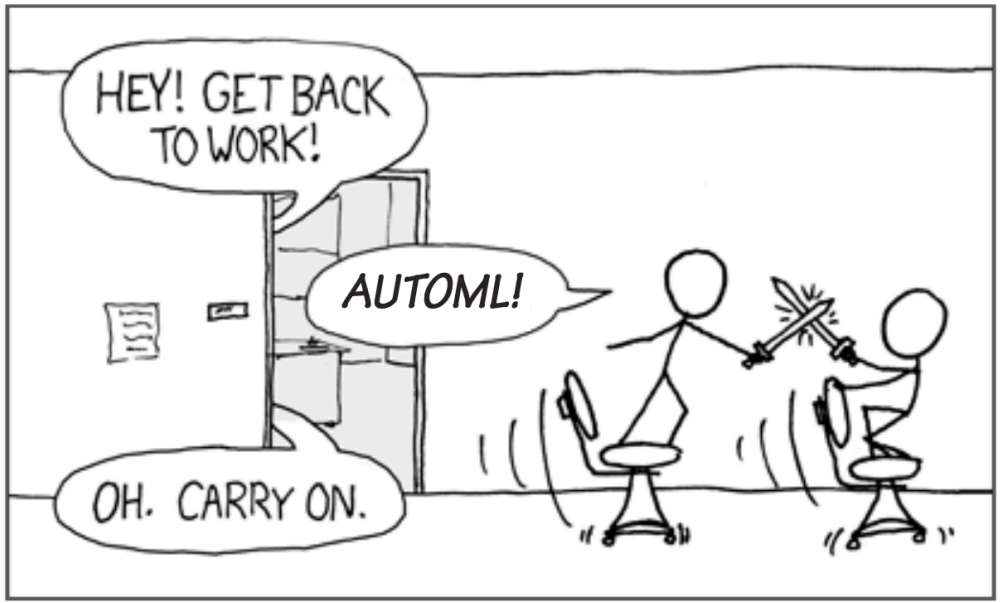
- Often needs tens of trials
  - Needs expertise to decide the next trial to explore
  - Time consuming

Automatic Machine Learning



- Focus on the application and data
- Needs to explore more trials
  - 10x more machine cost

THE DATA SCIENTIST'S #1 EXCUSE FOR LEGITIMATELY SLACKING OFF: "THE AUTOML TOOL IS OPTIMIZING MY MODELS!"



aws

Progress So Far

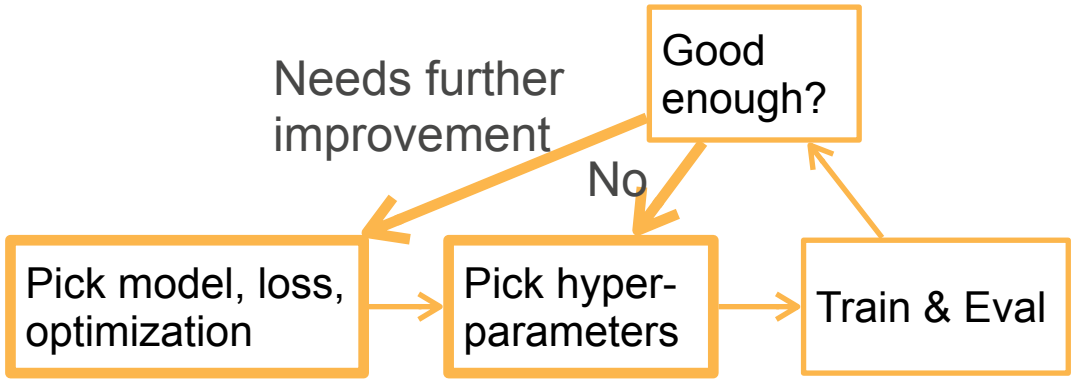
- ~10 trials for a data scientist for a task
- ~100 trials for AutoML to beat 50% data scientists
- ~1000 trials for AutoML to beat 99% data scientists
- For a normal size ML task in industrial
  - e.g: 1m users, 10K images

	Cost per trial	Cost per 100 trials on Cloud
xgboost/libsvm	5min on a CPU	$10
Deep neural networks	1hour on a GPU	$50

Cost of a data scientist per day: $500



Two Topics for Today



Hyper-parameter Optimization (HPO)



aws

Hyper-parameter Optimization (HPO)

- We first pick a ML model, a loss and an optimization method for training
- Then specify multiple hyper-paramaters during training, such as
  - Batch size, learning rate, weight decay, …
- HPO aims to automatically pick up the best set of value, or the best configuration

aws

Search Space

- We often specify candidates for each hyper-parameter
  - A list:
    batch_size in {16,32,64}
  - A linear/log/exp range:
    learning_rate in np.logspace(1e-6,1e-1,num=10)
  - Random values
    weight_decay = np.random.uniform(0, 1)
- The search space can be exponentially large
  - We can carefully design the space to improve efficiency

aws

Two Straight-forward Methods

- Grid search

```
For config in search_space:
        Train and eval with config
Pick the config with best eval results
```

  - Guarantee the best results
  - Only feasible for small problems

- Random search

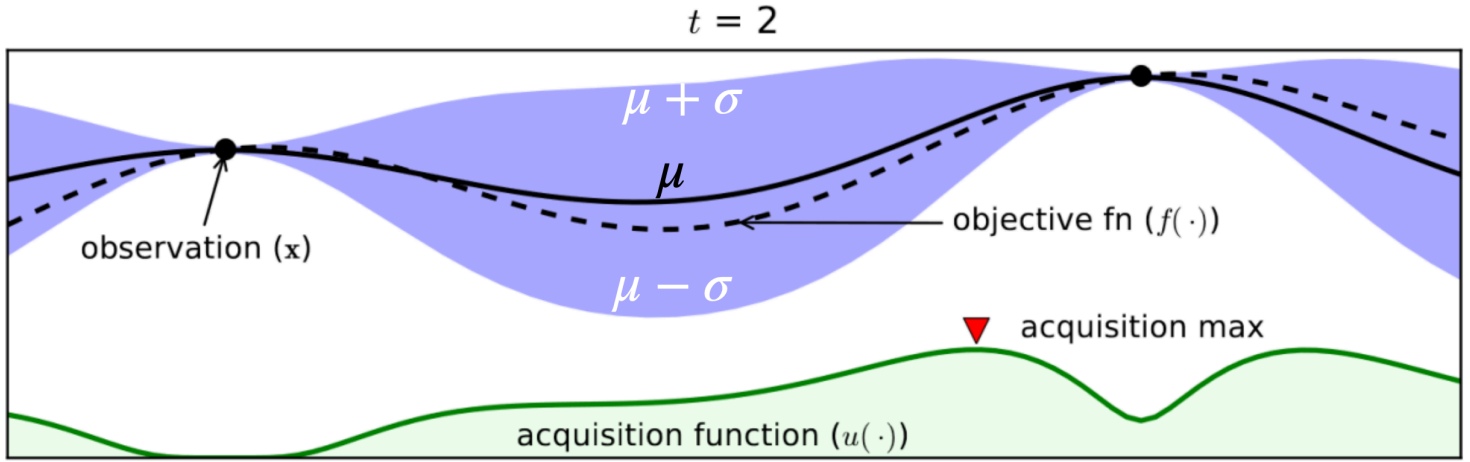
```
For _ in range(n)
        Randomly pick config from search_space
        Train and eval with config
Pick the config with best eval results
```

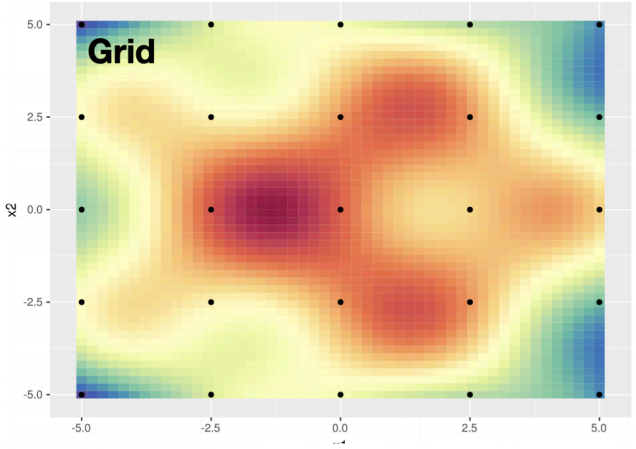
  - Simple and works pretty well in practice

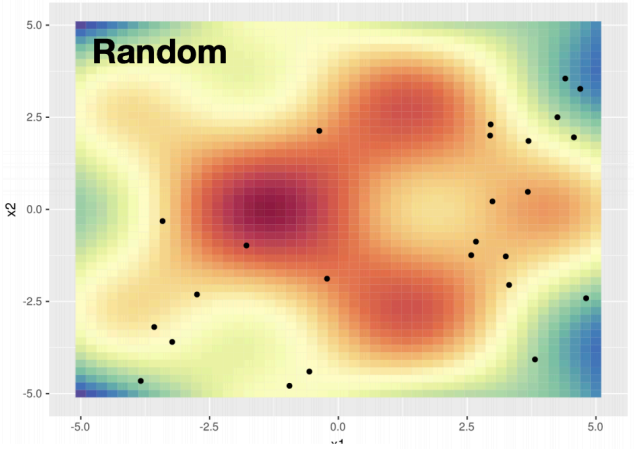
aws

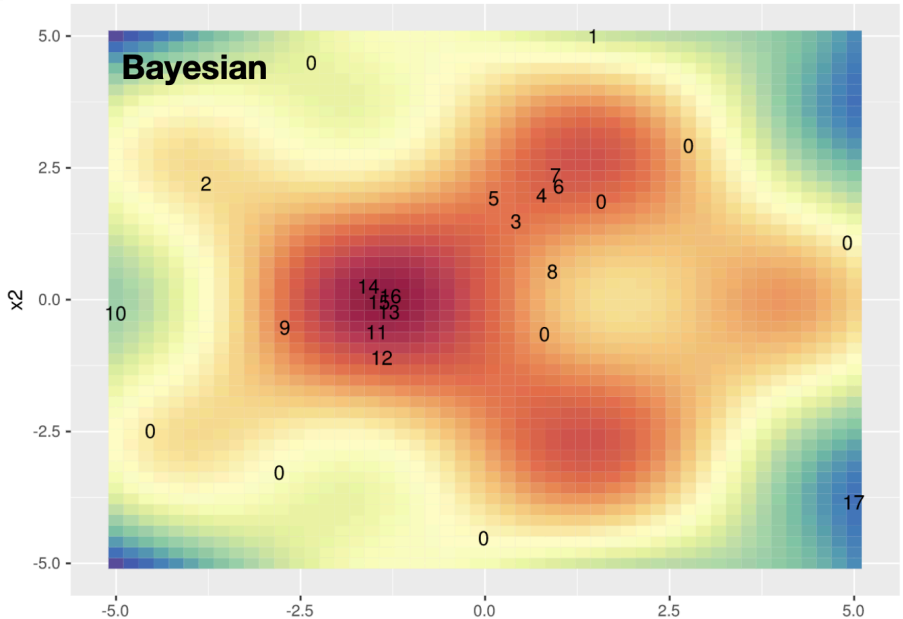
Bayesian Optimization (BO)

- Build a surrogate model to predict how well each configuration will work
  - Any probabilistic regression model: e.g. Gaussian processes
- Use an acquisition function to trade of exploration and exploitation, Sample the best configuration under that function









- Several choices of the surrogate function in BO
- BO works if we run enough trials to fit the model

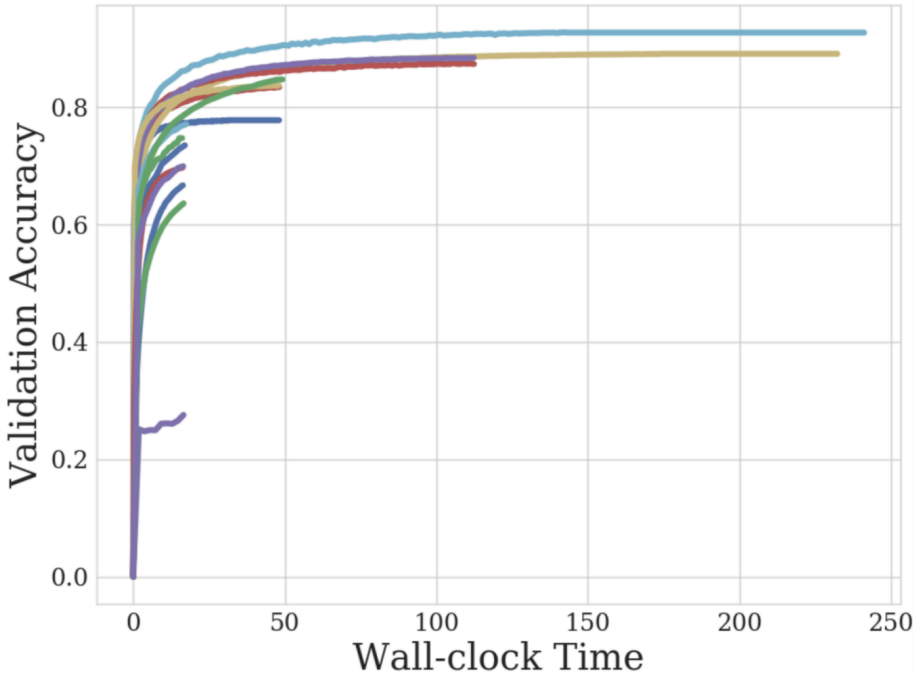
Black-box or White-box

- If a trial runs fast, e.g. 5min, we can finish a trial each time
- For deep neural network models, a trial needs 10-100 data epochs, we may stop bad trials earlier
  - Train on a subsampled datasets
  - Use less epochs
  - Reduce #layers, #channels

aws

Successive Halving (Jamieson, 1502.07943)

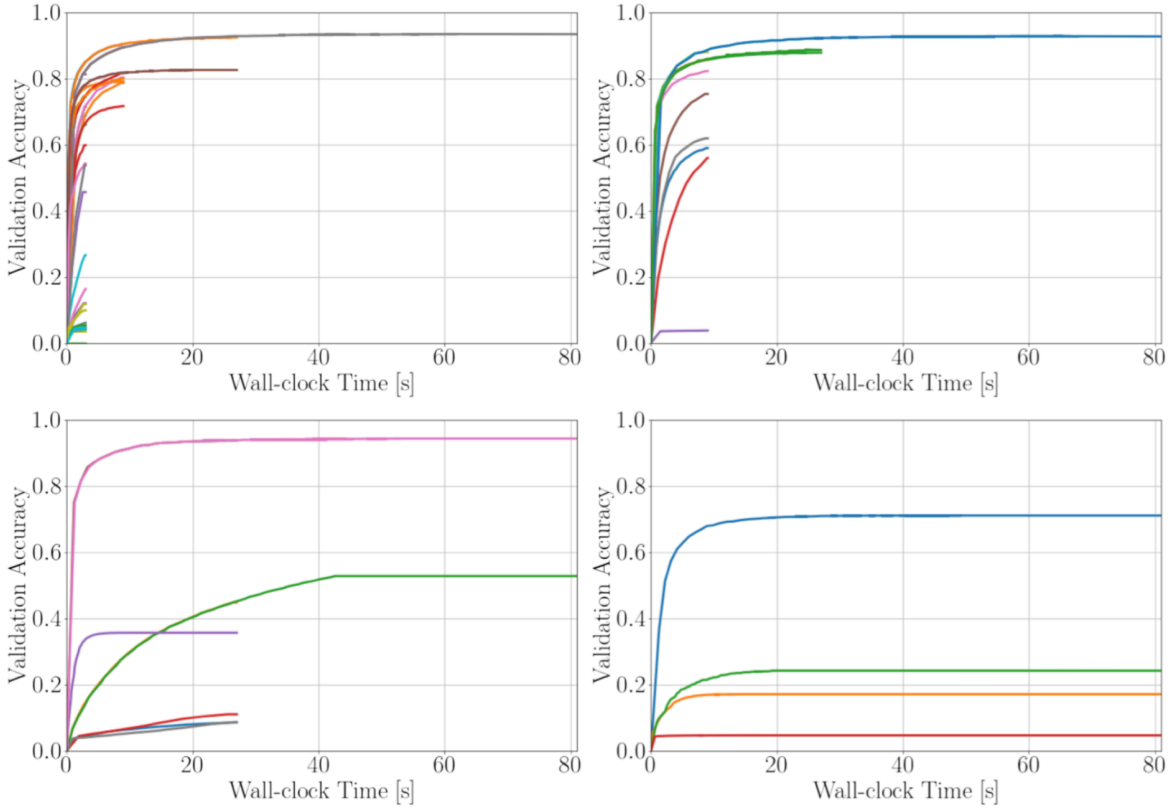
- Randomly pick *n* configurations to train *m* epochs
- Keep the best *n/2* configuration to train another *m* epochs
- Keep the best *n/4* configuration to train another *2m* epochs
- …
- *n* and *m* are chosen by the total budget #epoch needs for a full training



aws

Hyperband (Li, 1603.06560)

- In Successive Halving, n vs m is the explore/exploit trade-off
- Hyperband run multiple Successive Halving, each time decreases *n* but increases *m*



Recap

Model free	Bayesian optimization	Proxy
Grid search	surrogate/ acquisition function	Successive Halving
Random search		Hyperband

aws

Experiments on Object Detection

- Dataset

	#Images in dataset		
	Training	Validation	Testing
Clipart	400	100	500
Watercolor	1600	400	1000
Comic	1600	400	1000
VOC 2007	4009	1002	4952
VOC 2007+2012	13240	3311	4952

- Framework
  - Baseline: GluonCV https://gluon-cv.mxnet.io/
  - HPO: AutoGluon https://autogluon.mxnet.io/

aws

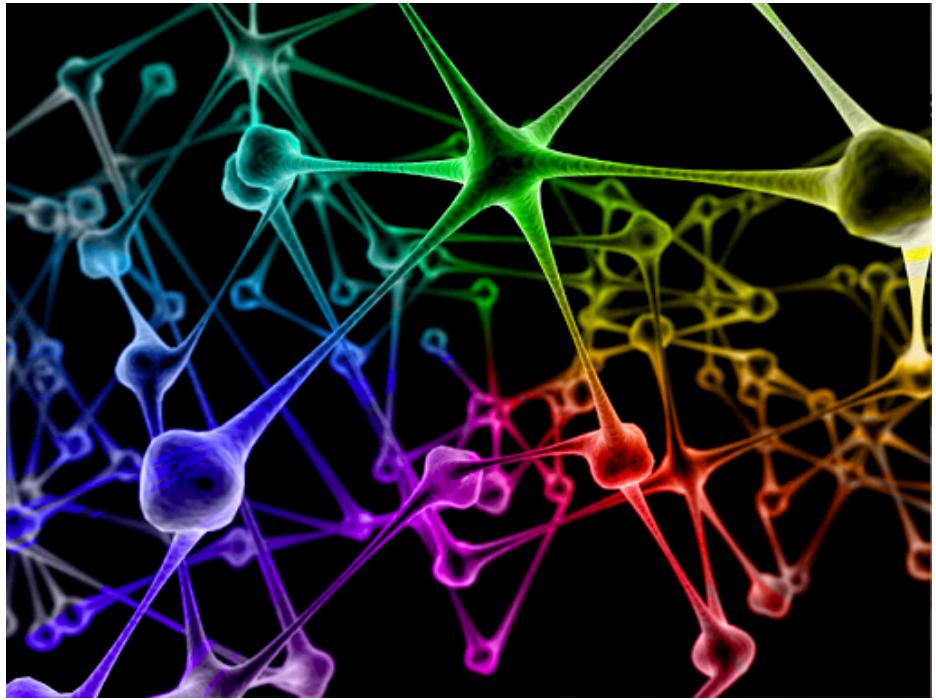
Results

- Running on AWS p3.16xl with 8 Nvidia v100

	GluonCV		AutoGluon	
	mAP	Time	mAP	Time
Clipart	0.513	0.8h	0.658	4.1h
Watercolor	0.915	1.3h	0.94	4.34
Comic	0.891	1.4h	0.936	4.7h
VOC2007	0.752	1.7h	0.85	6.2h * 3
VOC2007+2012	0.77	4.85h	0.86	15.2h * 3

aws

Neural-network Architecture Search (NAS)



Neural-network Architecture Search (NAS)

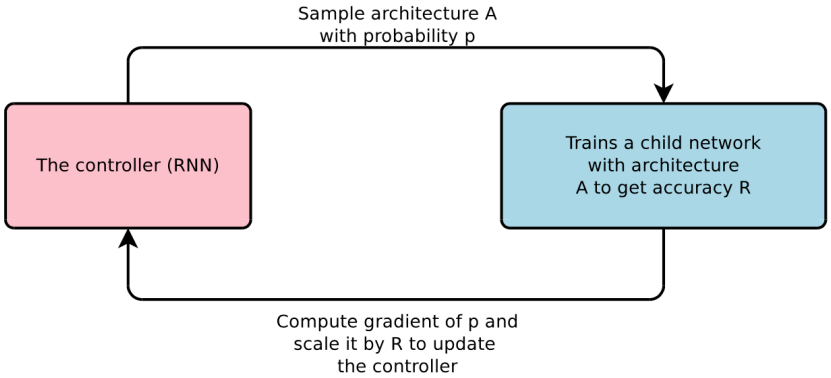
- A neural network have multiple hyper-parameters:
  - macro structure (resnet-like, mobilenet-like, #layers, …)
  - convolutional layer (kernel, #channels), dense layer (hidden size), recurrent layers, …
- NAS select a best neural network configuration from the search space
  - How to specify the search space
  - How to search

aws

NAS with Reinforcement Learning

- A recurrent network to generate network configurations
- A RL algorithm (REINFORCE) to train the recurrent network
- Very expensive, ~13K trials…

NAS (Zoph, 1611.01578)



- Follow-up works to improve efficiency
  - ENAS (Pham 1802.03268): force child networks to share weights

aws

The One-shot Approach

- Four steps:
  - Construct a single one-shot model presents a wide variety of architectures
  - Train the one-shot model
  - Evaluate candidate architectures using the pre-trained one-shot model
  - Re-train the most promising architecture from scratch
- Proxy metric: evaluate accuracy after several epochs
  - We only care about the candidate rankings

Differentiable Architecture Search

- Making architecture selecting differentiable
- Assume n layers in a block, make them fully connected
  - A architecture candidate has several path enabled
- DARTS (Liu, 1806.09055) propose to use softmax to enable/disable paths

$$x_j = \text{softmax}(\sum_{i<j} O_{i,j}(x_i))$$

x_i is the output of the i-th layer

aws

Improvements to DARTS

- ProxylessNAS (Cai, 1812.00332) binarize the Softmax scores in DARTS so only one path is active each time
  - memory efficient so we can train the one-shot network on the full dataset directly, no proxy metric is needed
- SNAS (Xie, 1812.09926) improves the deterministic DARTS by $x_j = \sum_{i<j} Z_{i,j} O_{i,j}(x_i)$, where $Z_{i,j}$ is a 0-1 random variable can be approximated by Gumbel softmax

NAS for Edge Devices

- There is a rising of running deep neural networks on edge devices
  - Edge devices are more and more powerful
  - Data privacy concerns
  - 5G makes users be more sensitive to latency
- Most neural networks are designed for server GPUs
- Edge device hardware is pretty diverse
  - CPU/GPU/TPU, 100x performance difference

aws

MnasNet - Platform-Aware NAS

- (Tan 1807.11626) Based on ENAS, use a RNN controller, and PPO algorithm
- Architecture is based on MobileNet V2
- Measure the latency on real hardware
  - Benchmark the latency for each operator, query them later
- The optimization goal is acc $\times$ [latency/T]^w, where $w = \alpha$ if latency <T, β otherwise

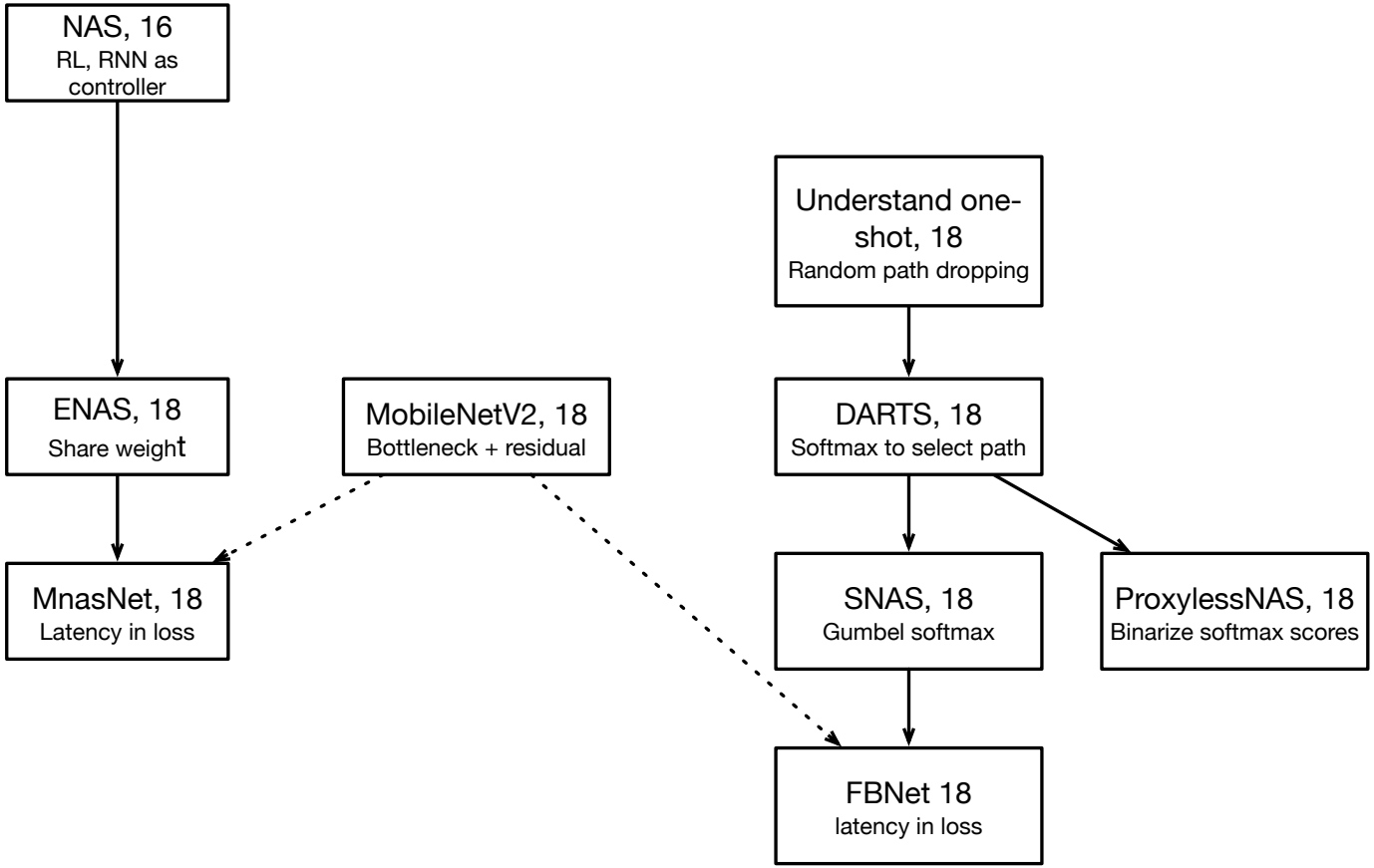
aws

FBNet - Hardware-Aware Differentiable NAS

- (Wu 1812.03443) Based on SNAS, but add latency to loss
- Similar to MnasNet
  - Architecture is based on MobileNet V2
  - Lookup real latency on hardware
- Cost function: cross_entropy $\times \log(\text{latency})^{\beta}$

aws

Recap


NAS, 16
RL, RNN as controller
ENAS, 18
Share weight
MnasNet, 18
Latency in loss
MobileNetV2, 18
Bottleneck + residual
Understand one-shot, 18
Random path dropping
DARTS, 18
Softmax to select path
SNAS, 18
Gumbel softmax
ProxylessNAS, 18
Binarize softmax scores
FBNet 18
latency in loss

Beyond HPO and NAS



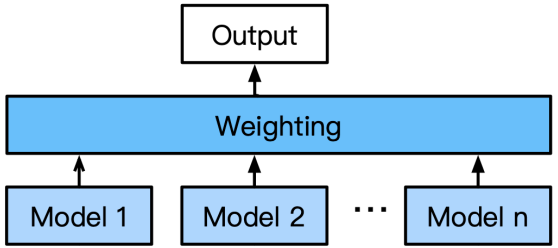
Other Approaches in AutoML

- AutoML: automatically find the best model within a given resource budget
- NAS and HPO are popular approaches, also hot research topics
- Meta-learning
  - For each trial, collect (task features, hyperparameters, results), then train a model to predict results
- Model ensembling
  - Use multiple models to boost accuracy, widely used in competitions (e.g. Kaggle)

aws

Model Ensembling

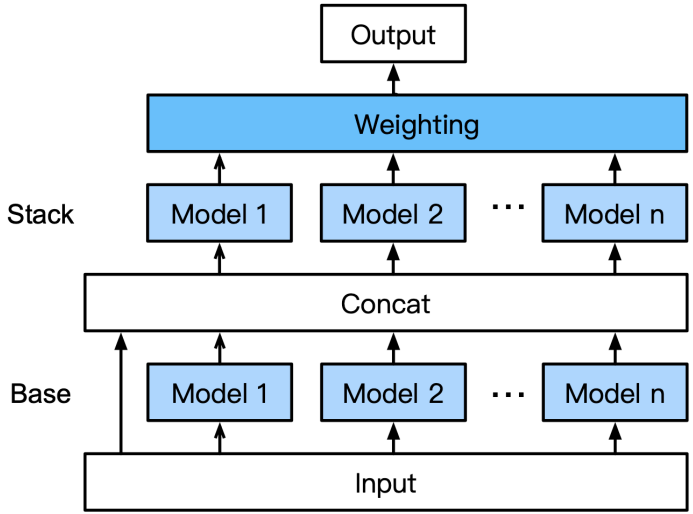
- Learn n different models on the same data
  - Same model with different hyper-parameters, or different models
- Combine the model predictions with weights
  - We can learn the weights



aws

Multi-layer stacking

- First learn an ensemble
- Generate a new dataset with the concatenated outputs of this ensemble
  - Can also include the original features
- Learn another ensemble
- May repeat by *k* times
  - 2-layer is a common choice



Experiments on Tabular Prediction

- Details in Erickson 2003.06505
- 39 datasets from OpenML AutoML benchmark
- 11 tabular datasets from Kaggle
- Run AutoGluon with

```
from autogluon import tabular as task
task.fit(train_data, label, eval_metric, time_limits=SEC, auto_stack=True)
```

No HPO (`hyperparameter_tune = True`), No NAS

aws

Comparing to other AutoML Frameworks

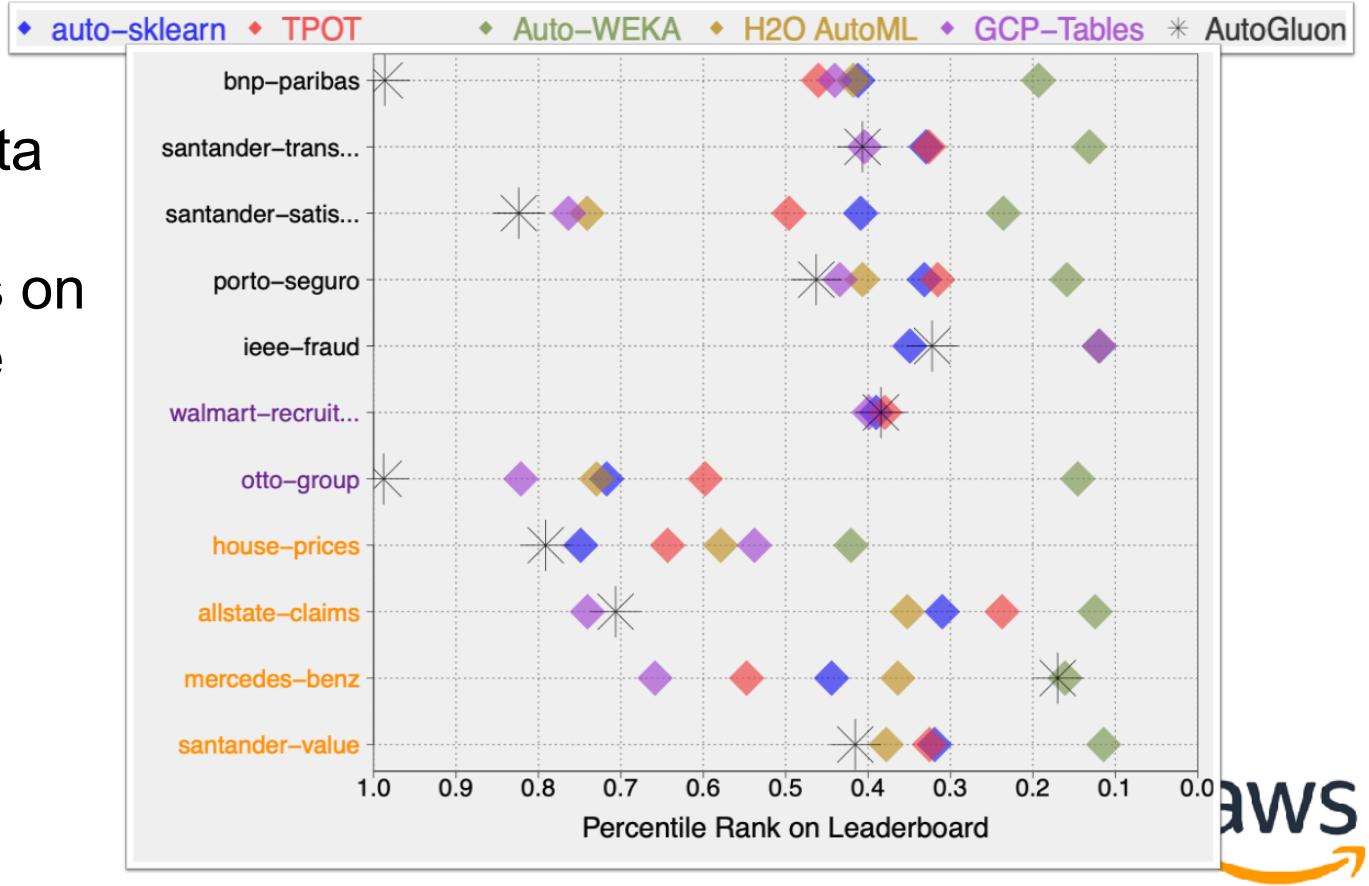
Framework	Wins	Losses	Failures	Champion	Avg. Rank	Avg. Rescaled Loss	Avg. Time (min)
AutoGluon	-	-	**1**	**23**	**1.8438**	**0.1385**	201
H2O AutoML	4	26	8	2	3.1250	0.2447	220
TPOT	6	27	5	5	3.3750	0.2034	235
GCP-Tables	5	20	14	4	3.7500	0.3336	**195**
auto-sklearn	6	27	6	3	3.8125	0.3197	240
Auto-WEKA	4	28	6	1	5.0938	0.8001	244

*GCP uses 92 instances

aws

Kaggle Results

- Beats most professional data scientists by running 4 hours on a CPU instance ($20)



Summary

- AutoML beats human on small/medium tasks with reasonable resource requirement
  - Will see a wide adoption in industrial
- Still time consuming for large-scale tasks
  - e.g. Train on ImageNet
- Solves new problems
  - Optimize model for given hardware

aws