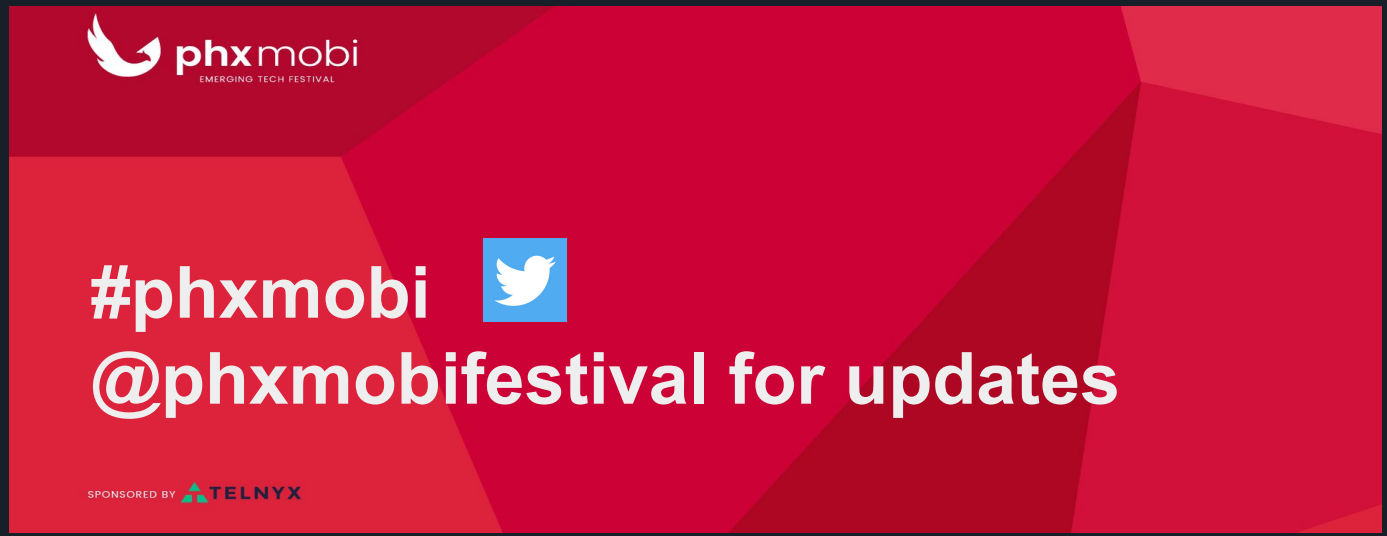
ABACUS.AI

AutoML: AI that build AI

Colin White, Abacus.AI

Slides (with hyperlinks): https://crwhite.ml



ABACUS.AI

Abacus.AI

AI for IT Operations

- Cloud Spend Alerts
- Early Incident Detection
- Predictive Maintenance

Recommender AI

- Personalized Recommendations
- Related Items
- Personalized Search
- Real-Time Feed Recommendations

Predictive Modeling

Marketing and Sales AI

- Predictive Lead Scoring
- Personalized Promotions
- Customer Churn Prediction
- Sales and Revenue Forecasting

Forecasting and Planning

- Demand Forecasting
- Real-Time Forecasting
- Financial Metrics Forecasting

Vision AI

- Image Classification & Detection
- Hybrid Models
- Image Segmentation

Fraud and Security

- Account Takeover and Defense
- Transaction/Credit Card Fraud
- Intelligent Threat Detection

Natural Language Processing

- Text extraction and classification
- NLP Powered Search
- Question & Answers
- Sentiment Analysis
- Hybrid Models

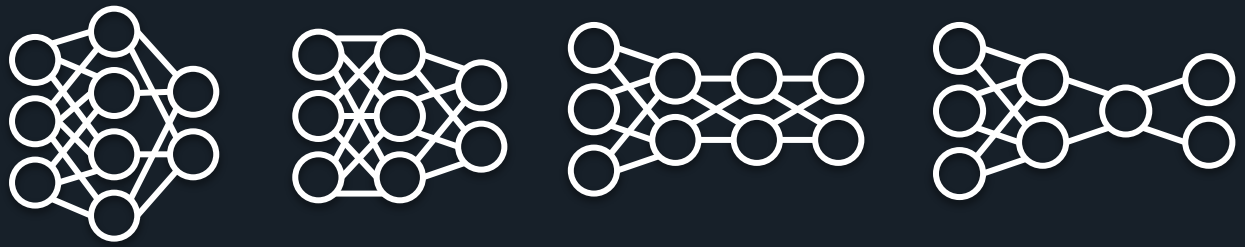
Anomaly Detection

- Timeseries Anomaly Detection
- Event Stream Anomaly Detection

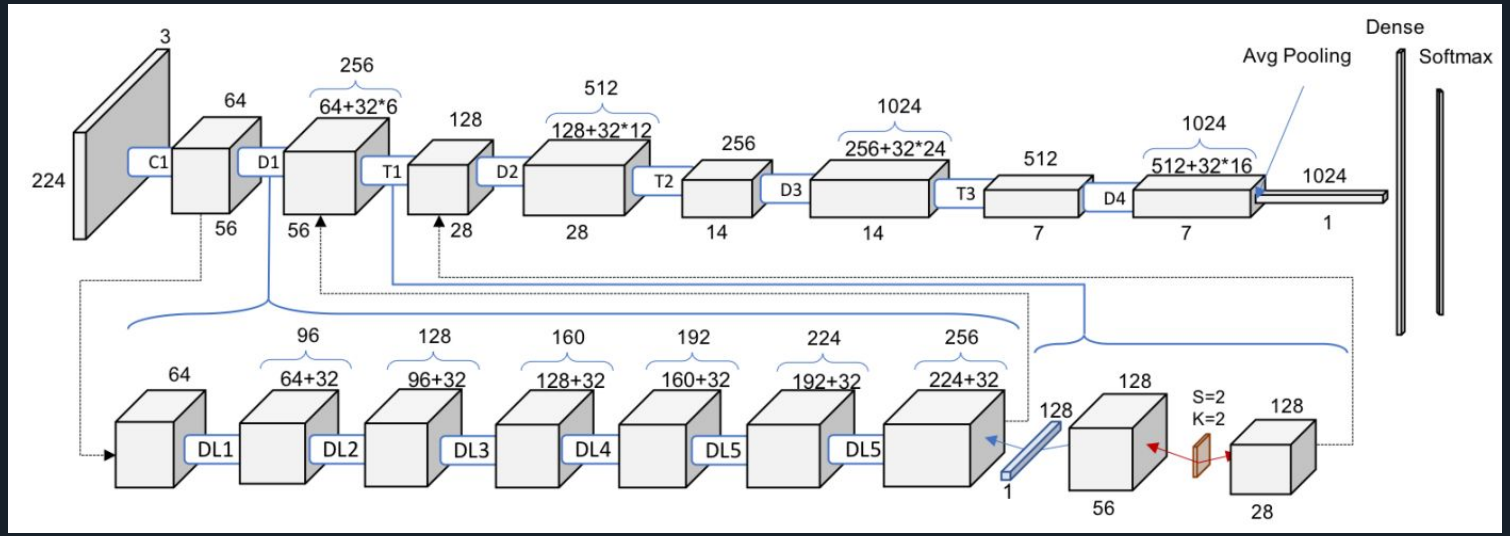
https://abacus.ai/publications

Deep learning

- Powerful machine learning technique
- Huge variety of neural networks for different tasks
- Becoming more specialized and complex
- Huge amount of hyperparameters



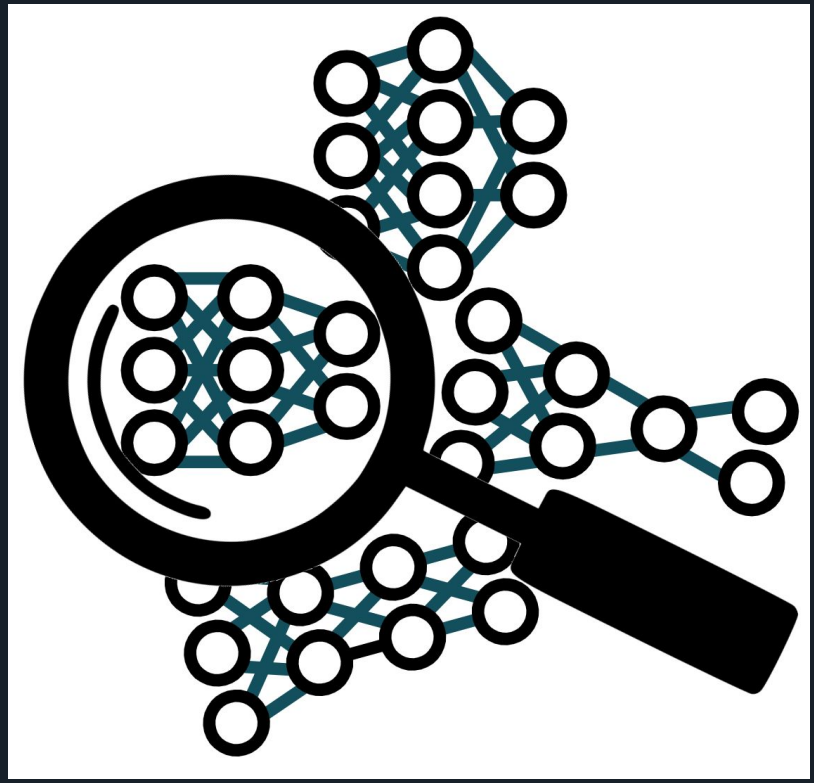
Deep learning



Accuracy on ImageNet has steadily improved over 10 years

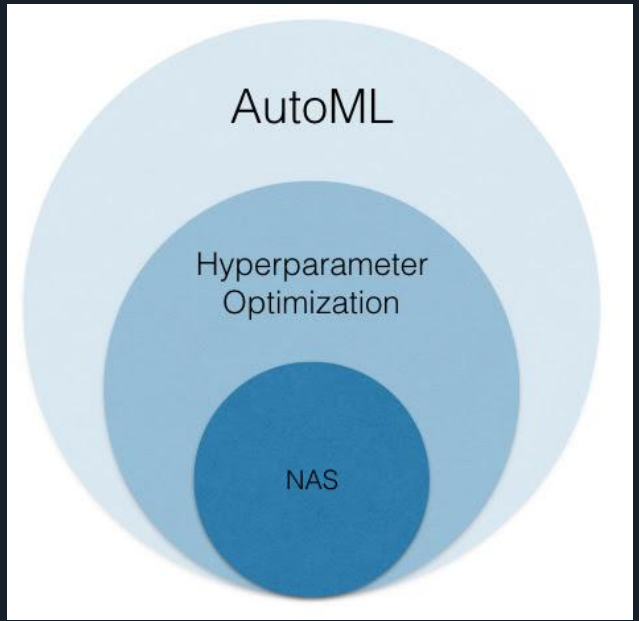
AutoML

- What if an algorithm could do this for us?
- AutoML (and neural architecture search) is a hot area of research
- Given a dataset, use an algorithm to find the best model for the dataset



Automated Machine Learning

- Automated machine learning
  - Data cleaning, model selection, HPO, NAS, ...
- Hyperparameter optimization (HPO)
  - Learning rate, dropout rate, batch size, ...
- Neural architecture search (NAS)
  - Finding the best neural architecture



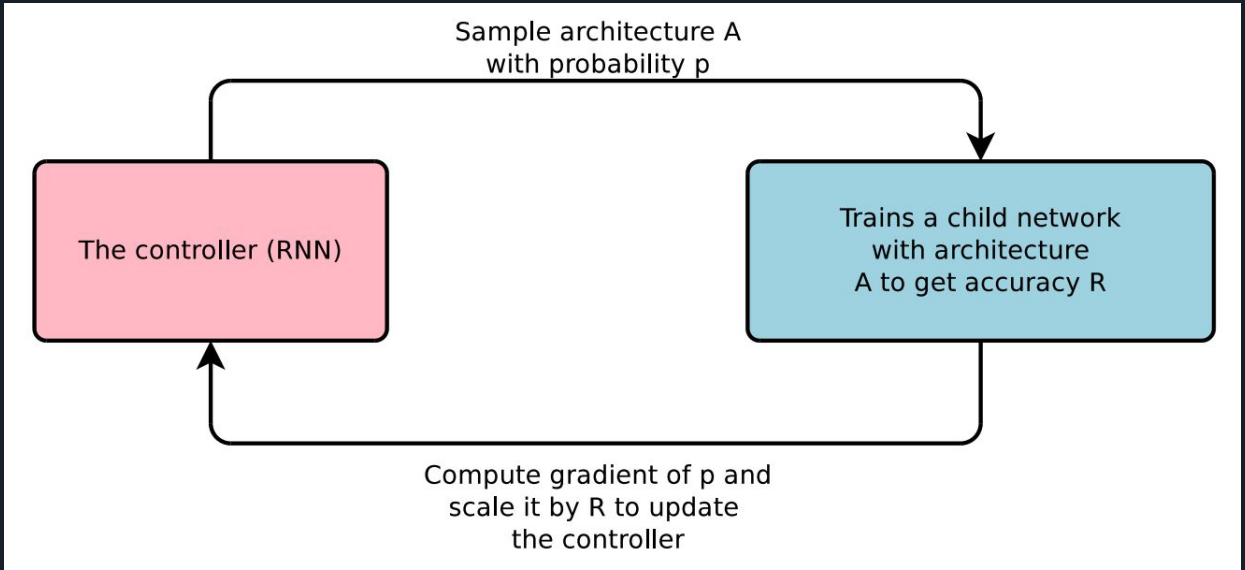
Source:https://determined.ai/blog/neural-architecture-search/

Outline

- Introduction to AutoML
- Performance Prediction
- Neural Architecture Search
- Architecture Encodings
- Conclusion

Performance prediction techniques

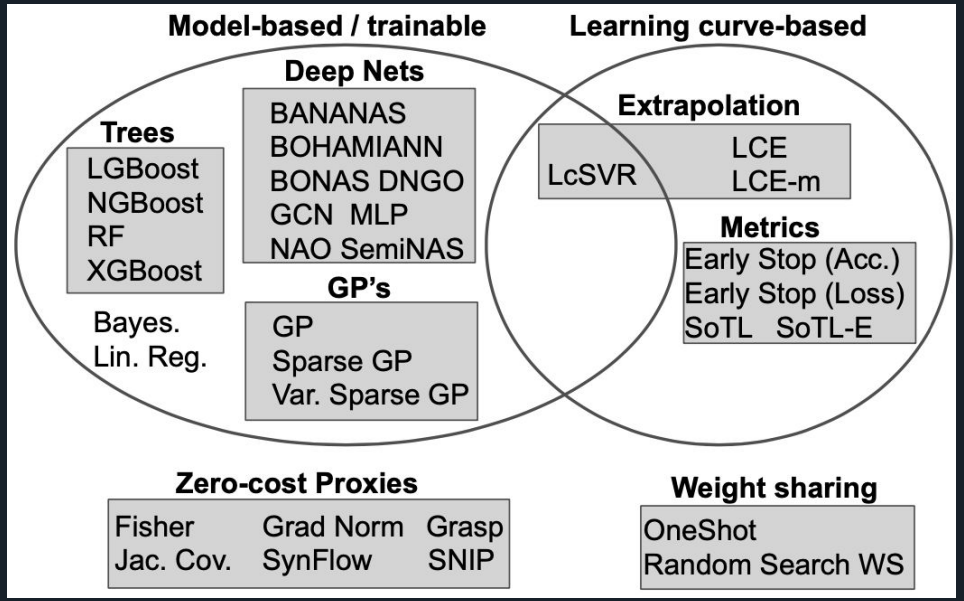
- Early NAS algos required fully training 1000s of architectures [Zoph and Le 2016]
- Recent algos use techniques to predict the final performance of architectures



Performance Predictors

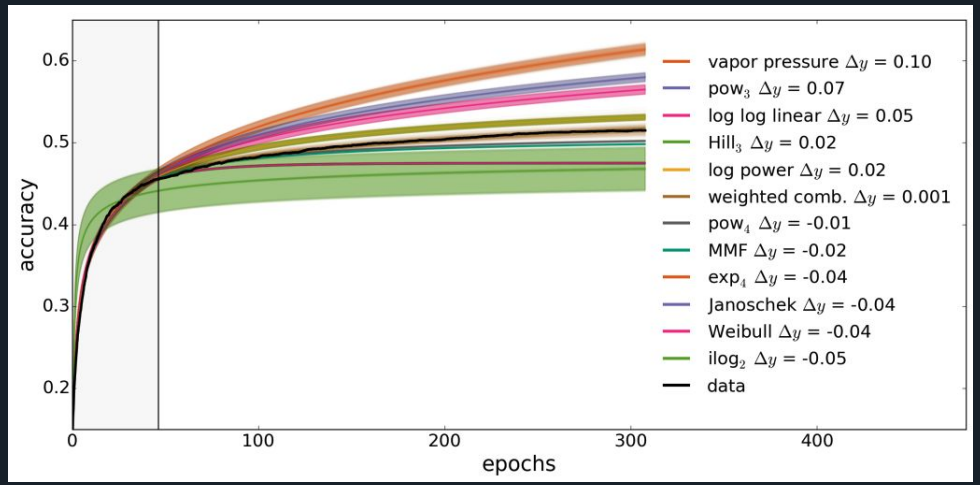
A *performance predictor* is any technique which predicts the final accuracy or ranking of architectures, without fully training them

- ***Initialization****:* pre-computation
- ***Query****:* architecture as input, output predicted accuracy



Learning curve based predictors

- Learning curve extrapolation
  - Fit partial learning curve to parametric model [Domhan et al. 2015]
  - Bayesian NN [Klein et al. 2017]
- Training statistics
  - Early stopping (val acc) [Elsken et al. 2018]
  - Sum of training losses [Ru et al. 2020]

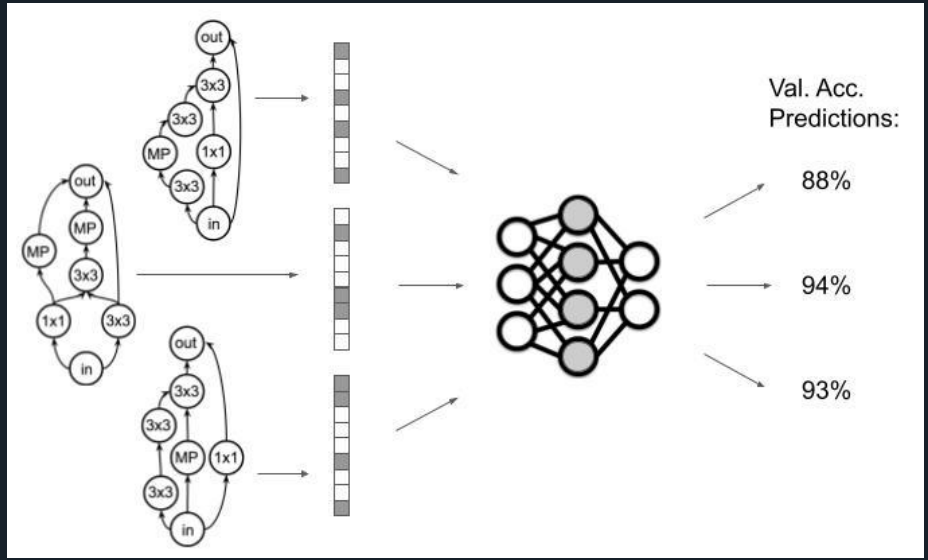


[Elsken et al. 2018]

No init time, high query time

Model-Based Predictors

- Supervised learning - regression
  - X - the architecture encoding (e.g. one-hot adjacency matrix)
  - Y - validation accuracy of trained architecture



[White et al. 2019]

- Gaussian processes [Kandasamy et al. 2018], [Jin et al. 2018]
- Boosted trees [Luo et al. 2020], [Siems et al. 2020]
- GNNs [Shi et al. 2019], [Wen et al. 2019]
- Specialized encodings [White et al. 2019], [Ning et al. 2020]

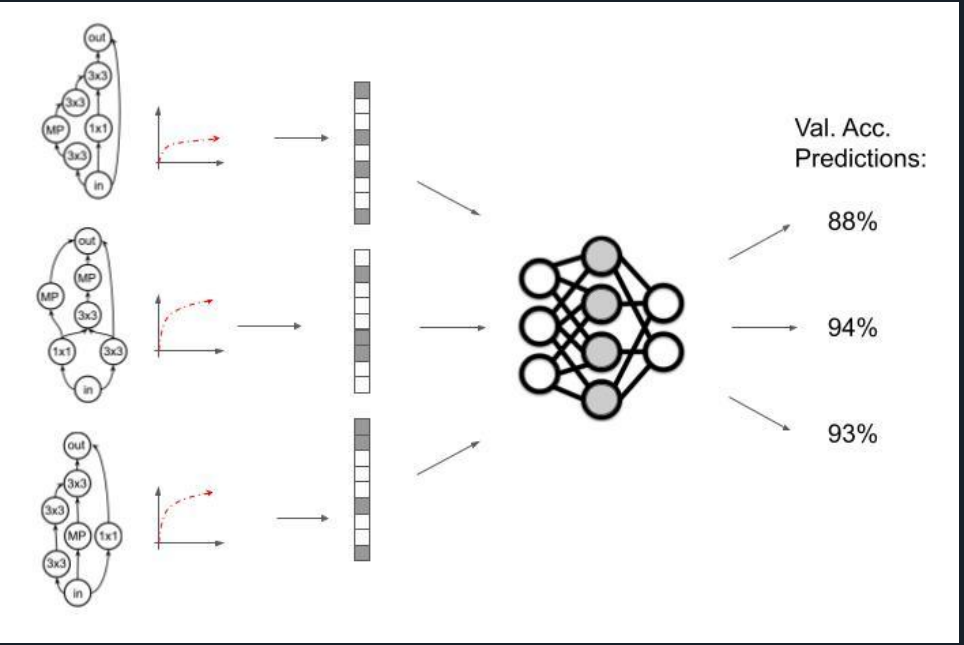
High init time, low query time

Hybrid model-based + LC predictors

Train a model, using partial learning curve + hyperparams, to predict final accuracy

- First and second derivatives as features, SVR [Baker et al. 2017]
- Full LC as features, Bayesian NN [Klein et al. 2017]

High init time, high query time

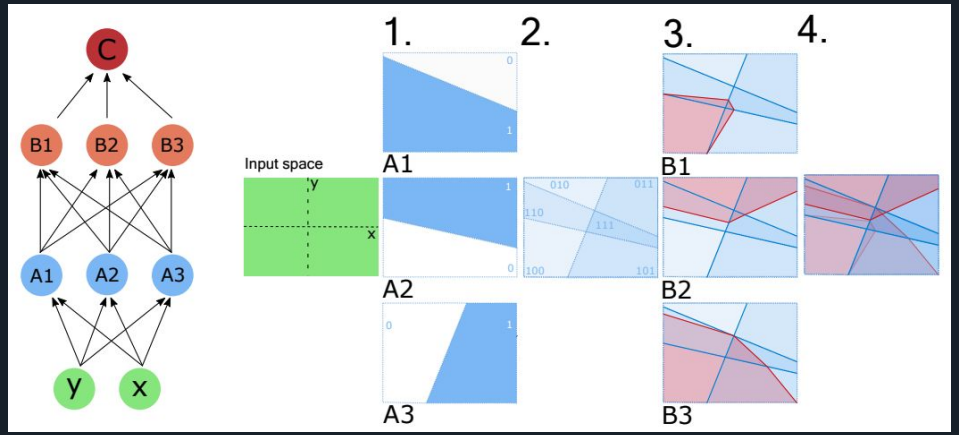


“Zero-cost” proxies

Compute a statistic of an architecture in 3-5 seconds

- Jacobian covariance [Mellor et al. 2020]
- Synaptic Flow [Abdelfattah et al. 2021]
  - SNIP [Lee et al. 2018]

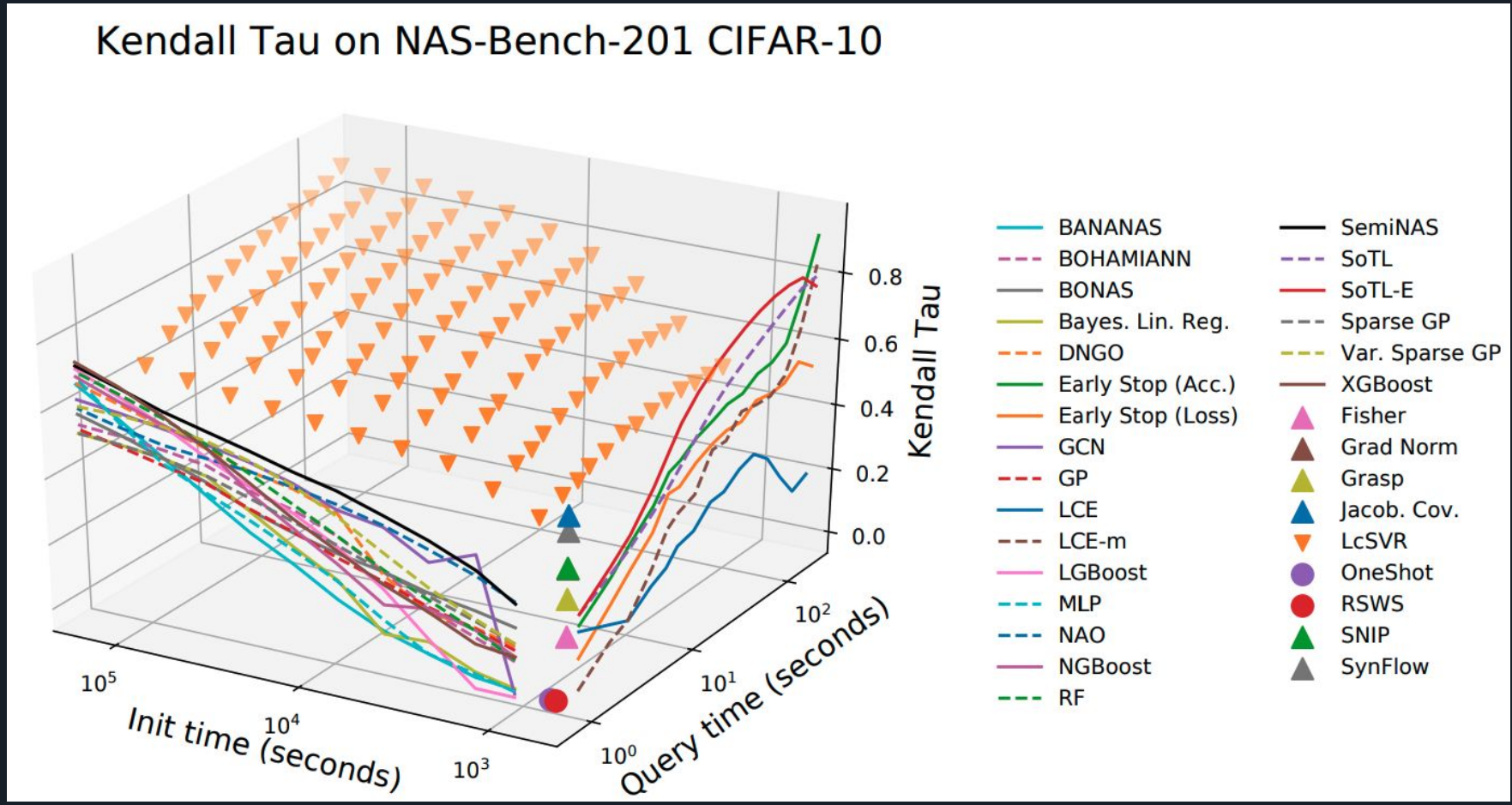
Low init time, low query time



[Mellor et al. 2020]

[Abdelfattah et al. 2021]

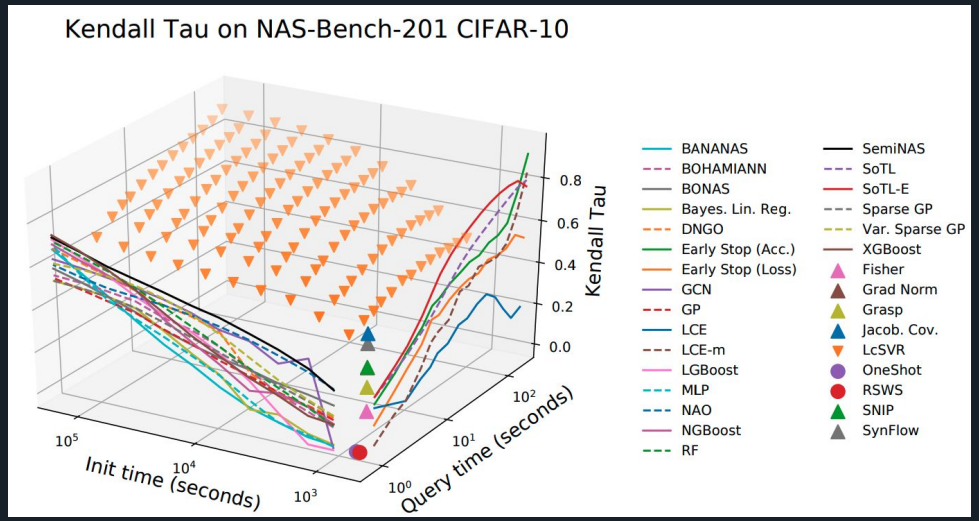
$$\texttt{snip}: \mathcal{S}_p(\theta) = \left|\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta\right|, \qquad \texttt{grasp}: \mathcal{S}_p(\theta) = -(H\frac{\partial \mathcal{L}}{\partial \theta}) \odot \theta, \qquad \texttt{synflow}: \mathcal{S}_p(\theta) = \frac{\partial \mathcal{L}}{\partial \theta} \odot \theta$$

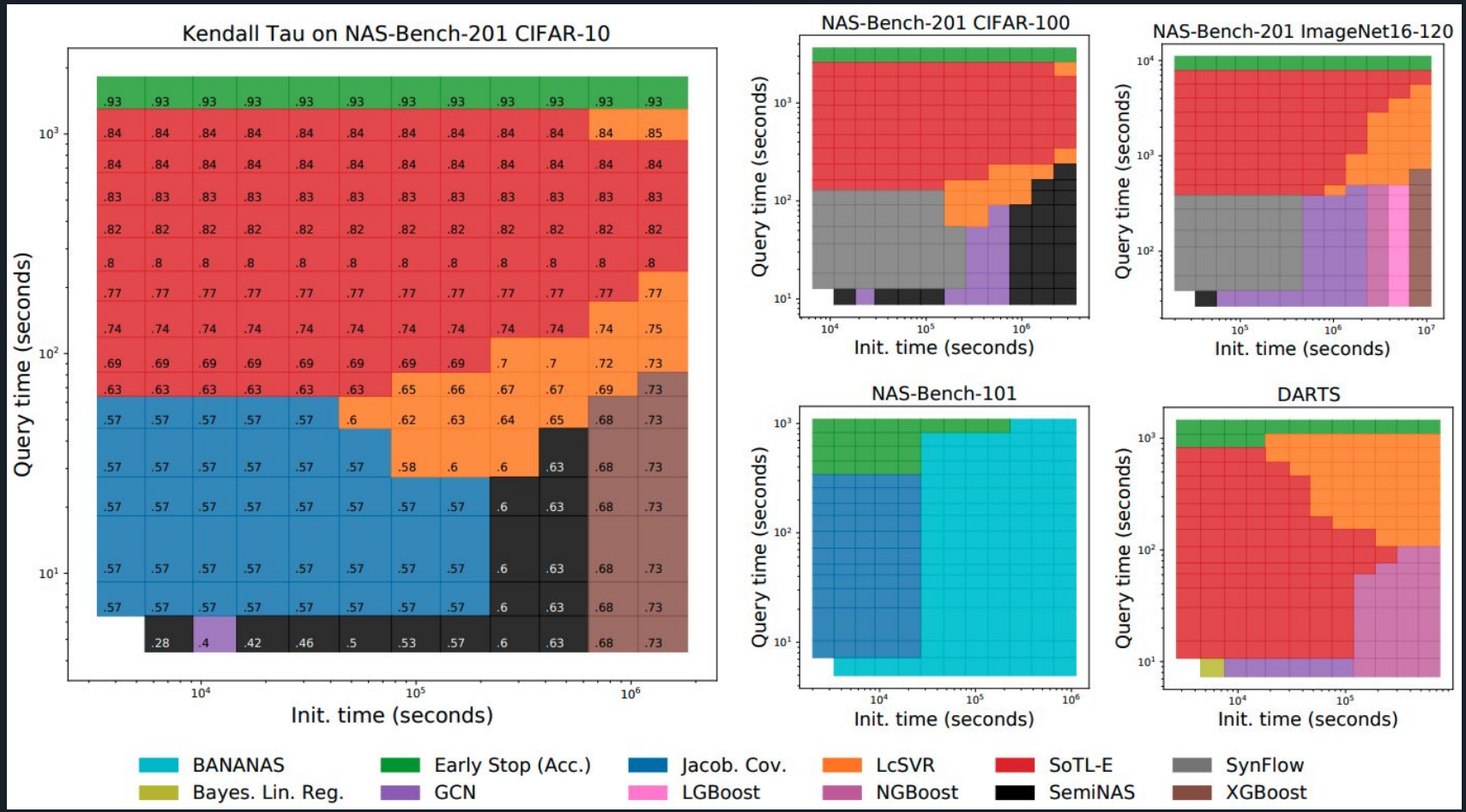

Kendall Tau on NAS-Bench-201 CIFAR-10
Kendall Tau
0.0
0.2
0.4
0.6
0.8
Init time (seconds)
10^5
10^4
10^3
Query time (seconds)
10^0
10^1
10^2
BANANAS
BOHAMIANN
BONAS
Bayes. Lin. Reg.
DNGO
Early Stop (Acc.)
Early Stop (Loss)
GCN
GP
LCE
LCE-m
LGBoost
MLP
NAO
NGBoost
RF
SemiNAS
SoTL
SoTL-E
Sparse GP
Var. Sparse GP
XGBoost
Fisher
Grad Norm
Grasp
Jacob. Cov.
LcSVR
OneShot
RSWS
SNIP
SynFlow

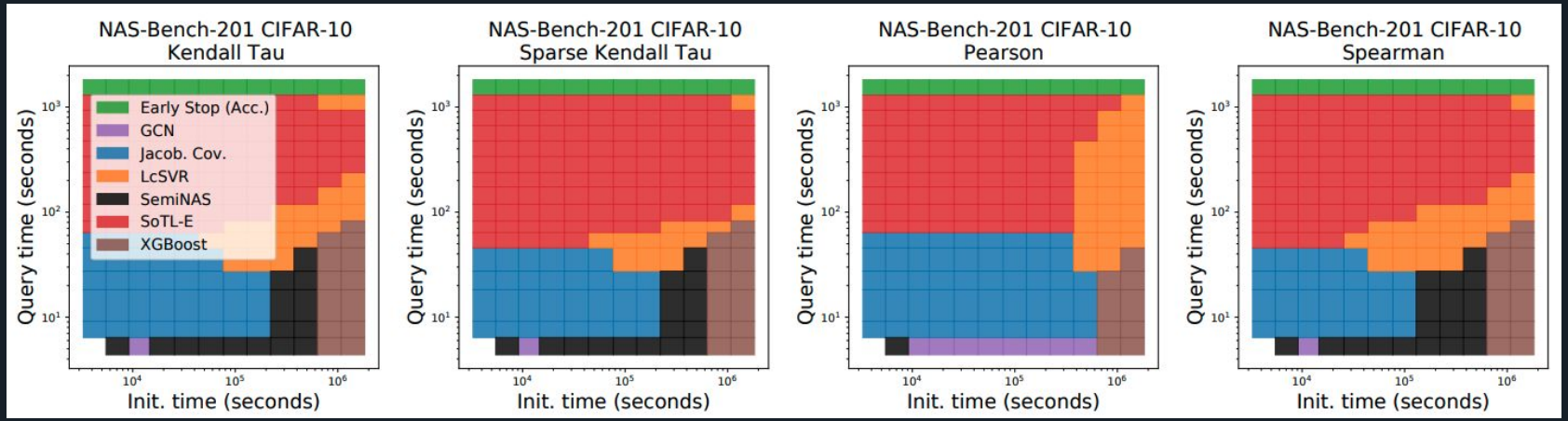
Notes on experiments

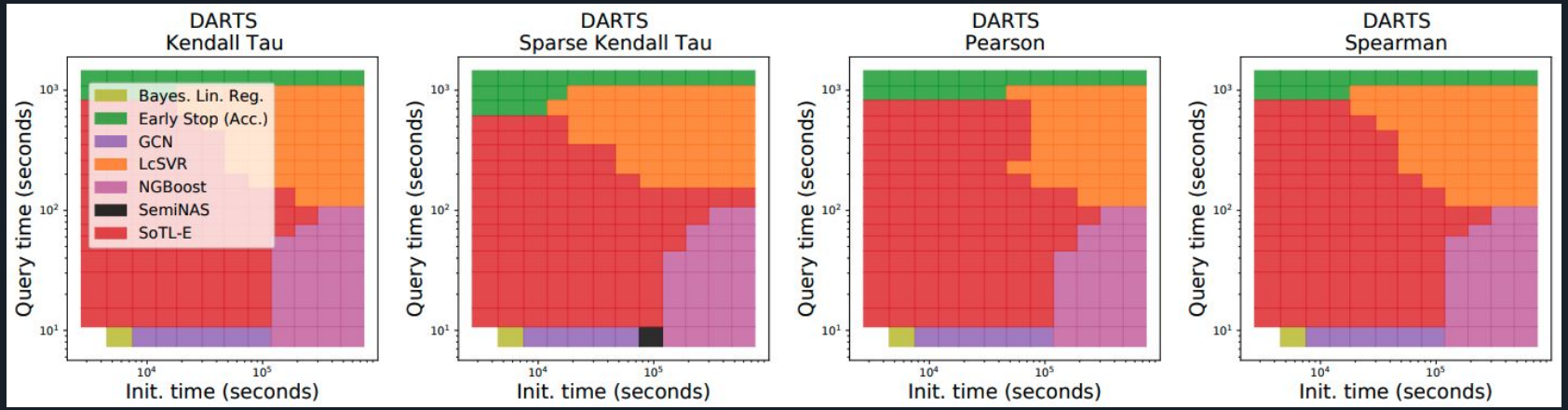
- Three axes of comparison: initialization time, query time, correlation / rank correlation metrics
- Train/test data drawn u.a.r.

- Light hyperparameter tuning
  - Levels the playing field
  - Cross-validation is often used during NAS



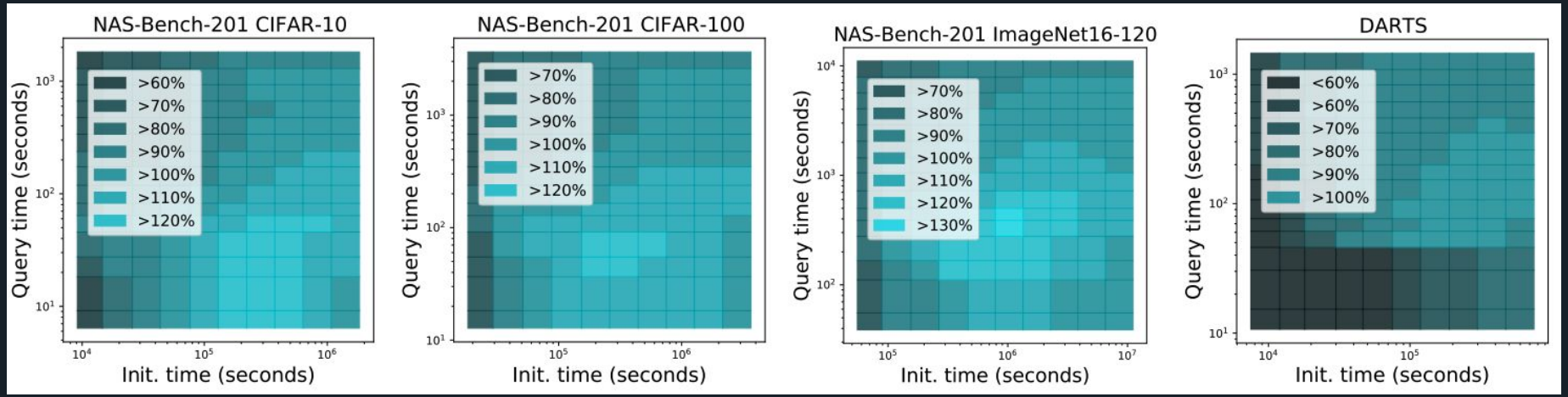

Kendall Tau on NAS-Bench-201 CIFAR-10
Query time (seconds)
Init. time (seconds)
NAS-Bench-201 CIFAR-100
NAS-Bench-201 ImageNet16-120
NAS-Bench-101
DARTS
BANANAS
Bayes. Lin. Reg.
Early Stop (Acc.)
GCN
Jacob. Cov.
LGBoost
LcSVR
NGBoost
SoTL-E
SemiNAS
SynFlow
XGBoost


NAS-Bench-201 CIFAR-10
Kendall Tau
NAS-Bench-201 CIFAR-10
Sparse Kendall Tau
NAS-Bench-201 CIFAR-10
Pearson
NAS-Bench-201 CIFAR-10
Spearman
Early Stop (Acc.)
GCN
Jacob. Cov.
LcSVR
SemiNAS
SoTL-E
XGBoost
Query time (seconds)
Init. time (seconds)

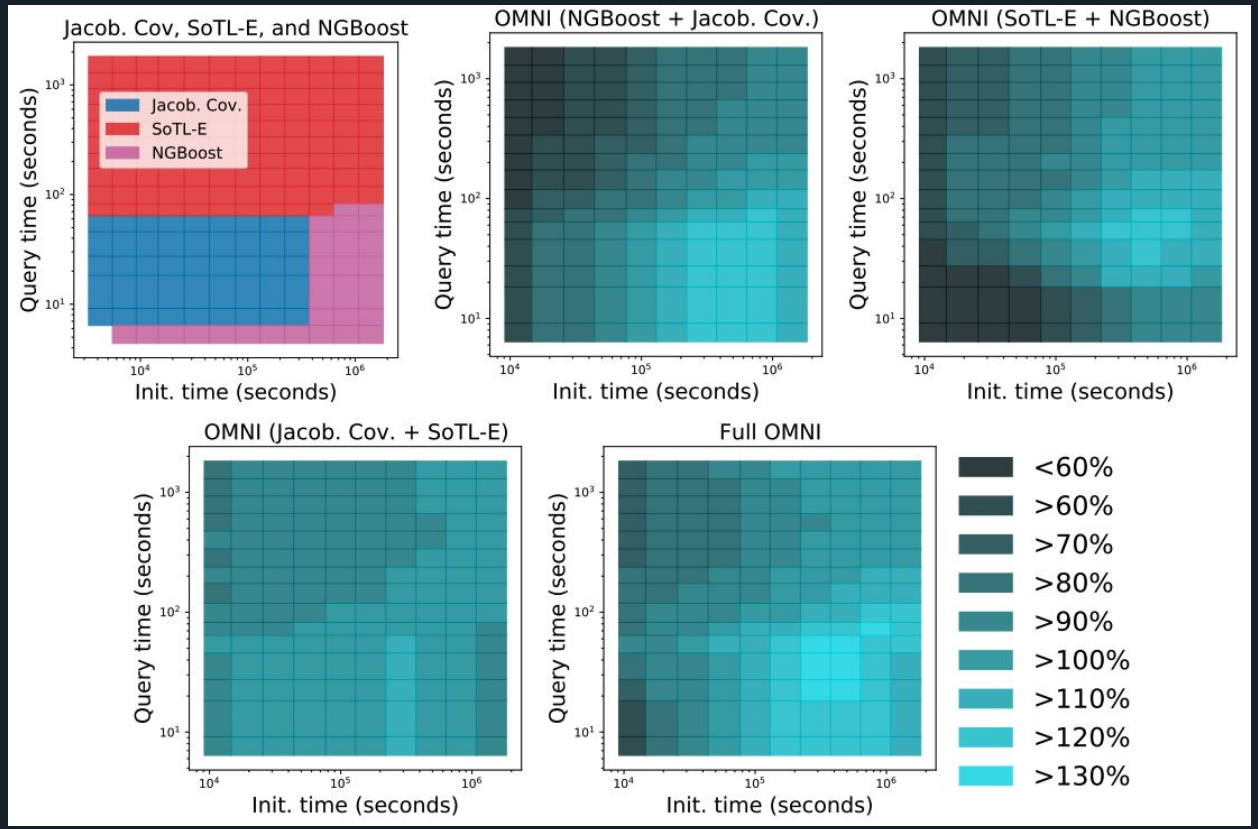

DARTS
Kendall Tau
DARTS
Sparse Kendall Tau
DARTS
Pearson
DARTS
Spearman
Bayes. Lin. Reg.
Early Stop (Acc.)
GCN
LcSVR
NGBoost
SemiNAS
SoTL-E
Query time (seconds)
Init. time (seconds)

OMNI: The Omnipotent Predictor

- Combine best predictors from three families: SoTL + Jacob. Cov + NGBoost
- Consistent performance almost everywhere
- 20% improvement in most-competitive bottom row

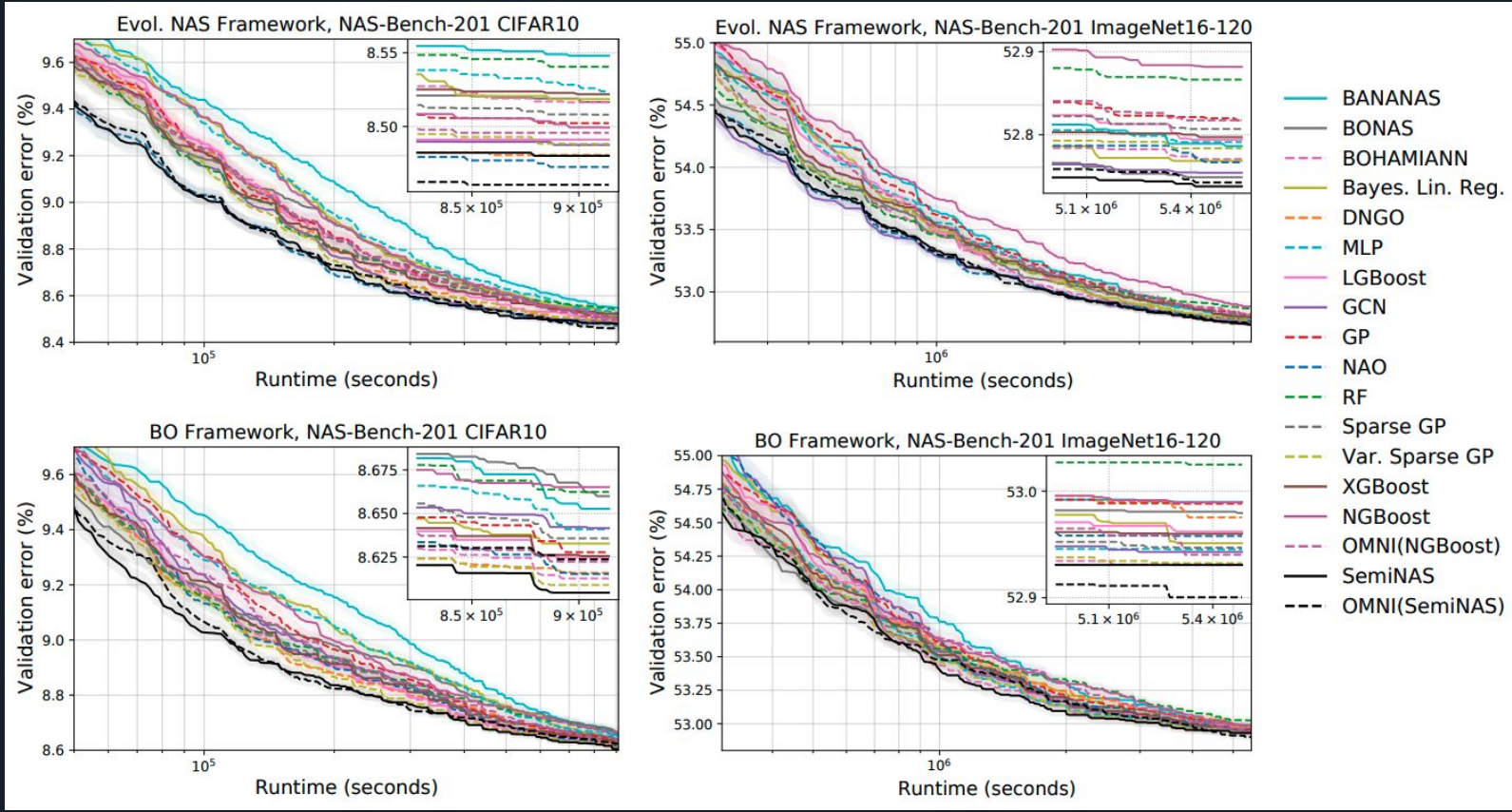


OMNI Ablation



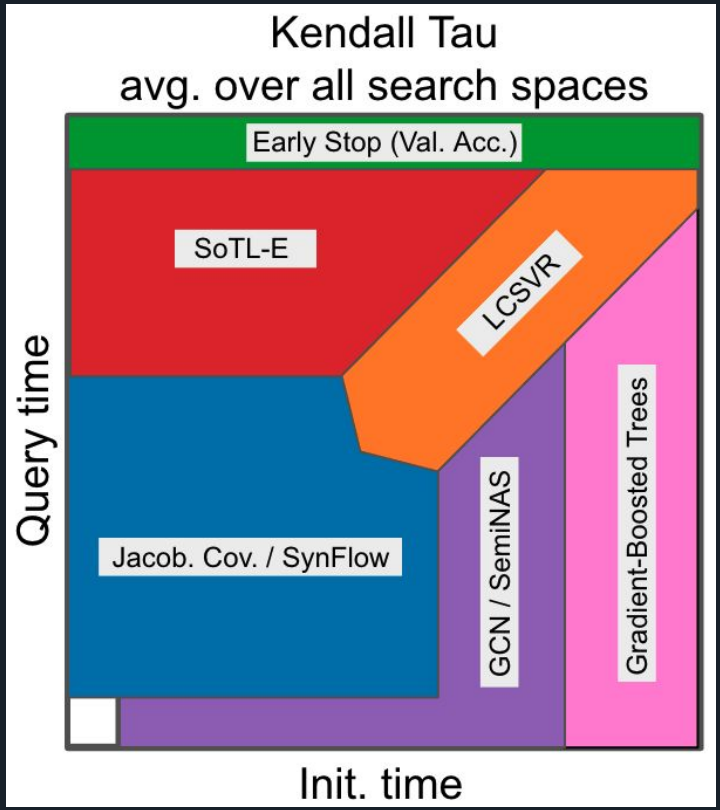
- Jacob. Cov + SoTL-E is consistent
- NGBoost needed for top performance in lower middle/right

NAS Experiments



So… How powerful are performance predictors?

- Largely the same trends across all experiments
- Combining predictors works the best

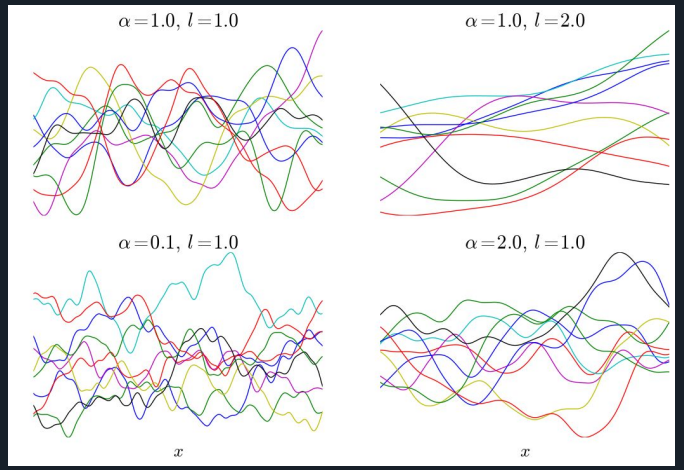


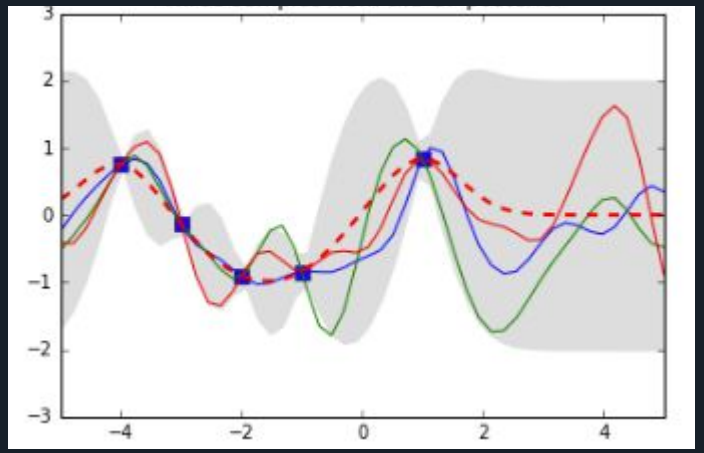
Outline

- Introduction to AutoML
- Performance Prediction
- Neural Architecture Search
- Architecture Encodings
- Conclusion

Bayesian optimization

- NASBOT [Kandasamy et al. ‘18], Auto-Keras [Jin et al. ‘18]
- Popular method in HPO, but not straightforward for NAS
  - Gaussian process - scalability
  - Hand-designed *distance function*

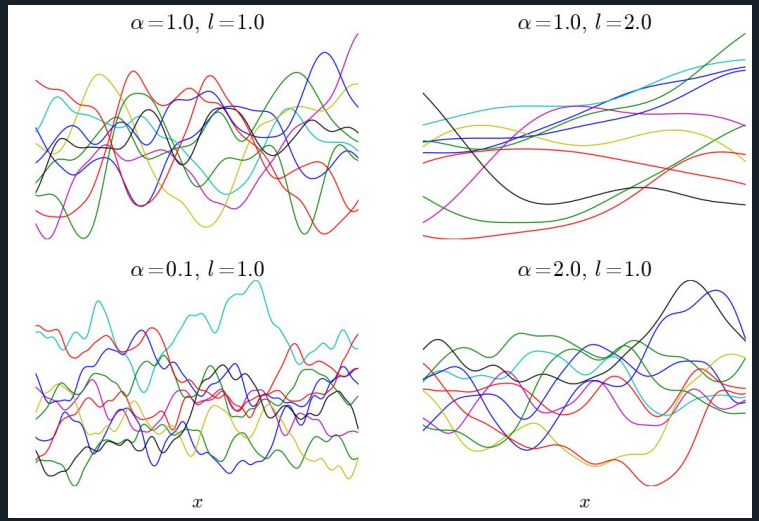


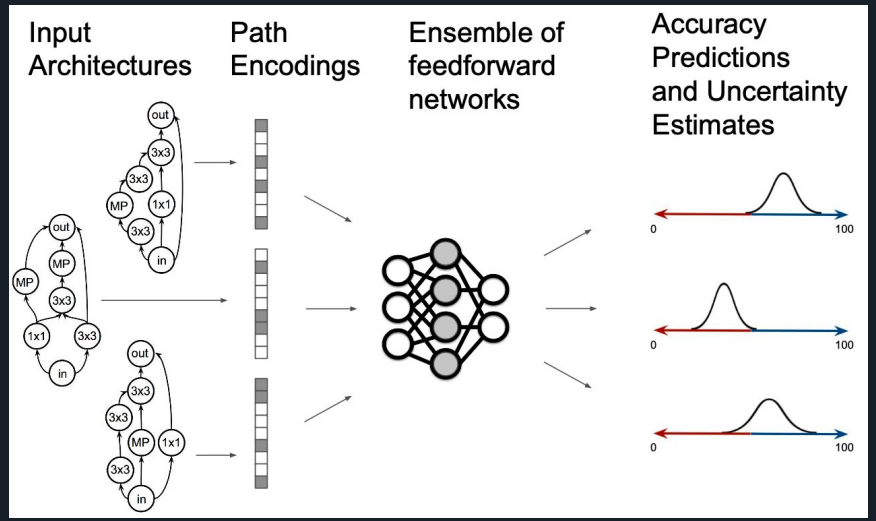


Source: http://keyonvafa.com/gp-tutorial/, https://katbailey.github.io/post/gaussian-processes-for-dummies/

“BO + Neural Predictor” Framework

- NASGBO [Ma et al. ‘19], BONAS [Shi et al. ‘19], BANANAS





Gaussian process

Neural predictor

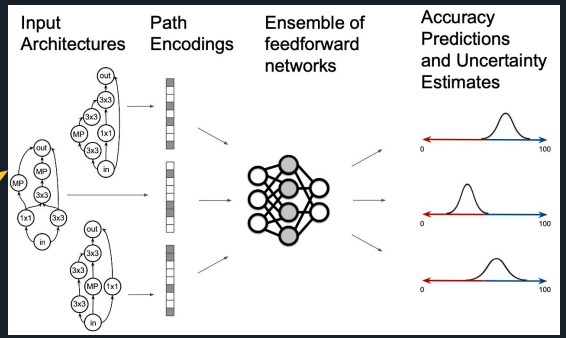
“BO + Neural Predictor” Framework

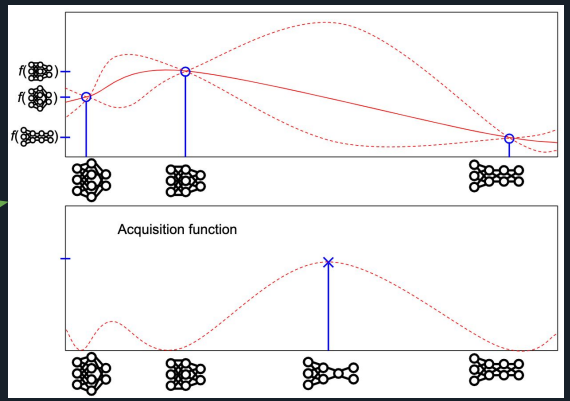
Algorithm 1 BANANAS

Input: Search space A, dataset D, parameters t_0, T, M, c, x, acquisition function ϕ, function $f(a)$ returning validation error of a after training.

1. Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from A and train them on D.
2. For t from t_0 to T,
   i. Train an ensemble of meta neural networks on $\{(a_0, f(a_0)), \ldots, (a_t, f(a_t))\}$.
   ii. Generate a set of c candidate architectures from A by randomly mutating the x architectures a from $\{a_0, \ldots, a_t\}$ that have the lowest value of $f(a)$.
   iii. For each candidate architecture a, evaluate the acquisition function $\phi(a)$.
   iv. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$, and evaluate $f(a_{t+1})$.

Output: $a^* = \text{argmin}_{t=0,\ldots,T} f(a_t)$.





Train 10 arch.’s each iteration

“BO + Neural Predictor” Components

Algorithm 1 BANANAS

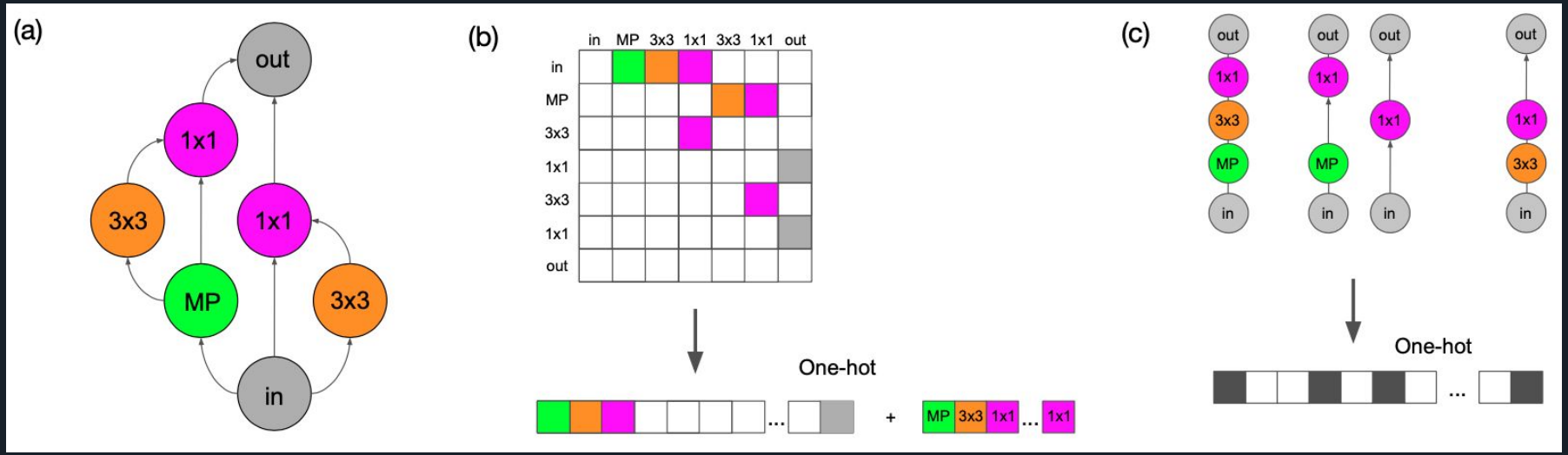
Input: Search space A, dataset D, parameters t_0, T, M, c, x, acquisition function ϕ, function $f(a)$ returning validation error of a after training.

1. Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from A and train them on D.
2. For t from t_0 to T,
   i. Train an ensemble of meta neural networks on $\{(a_0, f(a_0)), \ldots, (a_t, f(a_t))\}$.
   ii. Generate a set of c candidate architectures from A by randomly mutating the x architectures a from $\{a_0, \ldots, a_t\}$ that have the lowest value of $f(a)$.
   iii. For each candidate architecture a, evaluate the acquisition function $\phi(a)$.
   iv. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$, and evaluate $f(a_{t+1})$.

Output: $a^* = \text{argmin}_{t=0,\ldots,T} f(a_t)$.

- Architecture encoding
- Uncertainty calibration
- Neural predictor architecture
- Acquisition function
- Acquisition optimization strategy

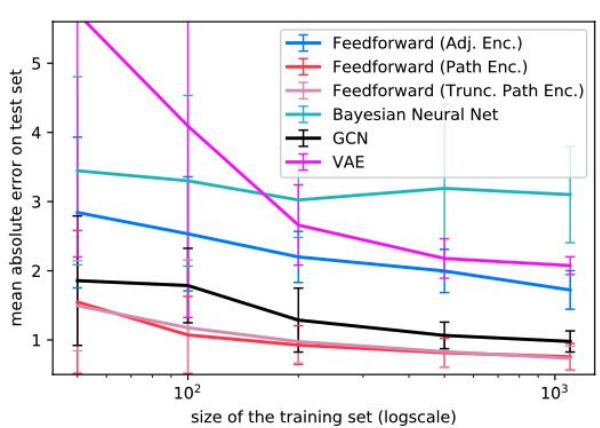
Architecture Encodings

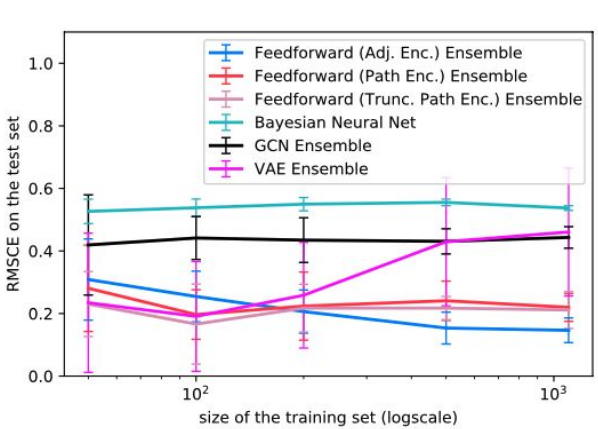


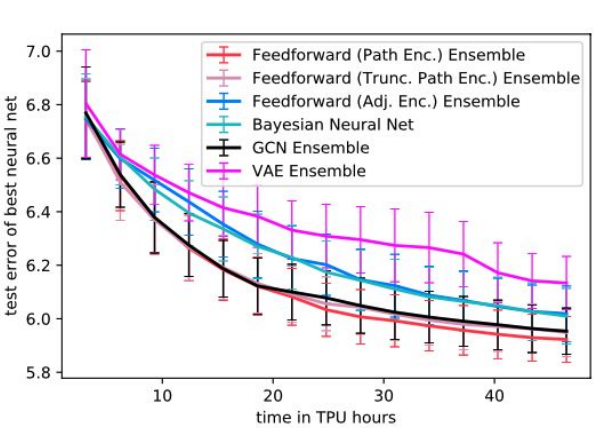
- Path encoding: each path from input to output is a feature

ABACUS.AI

Uncertainty prediction + architecture





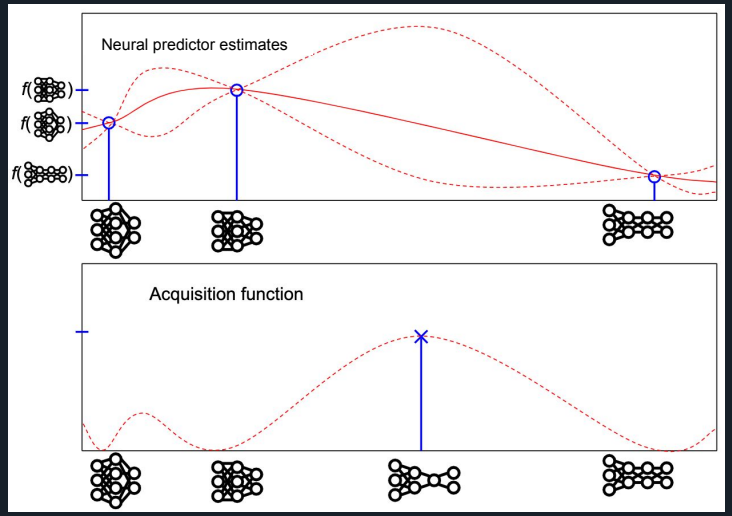


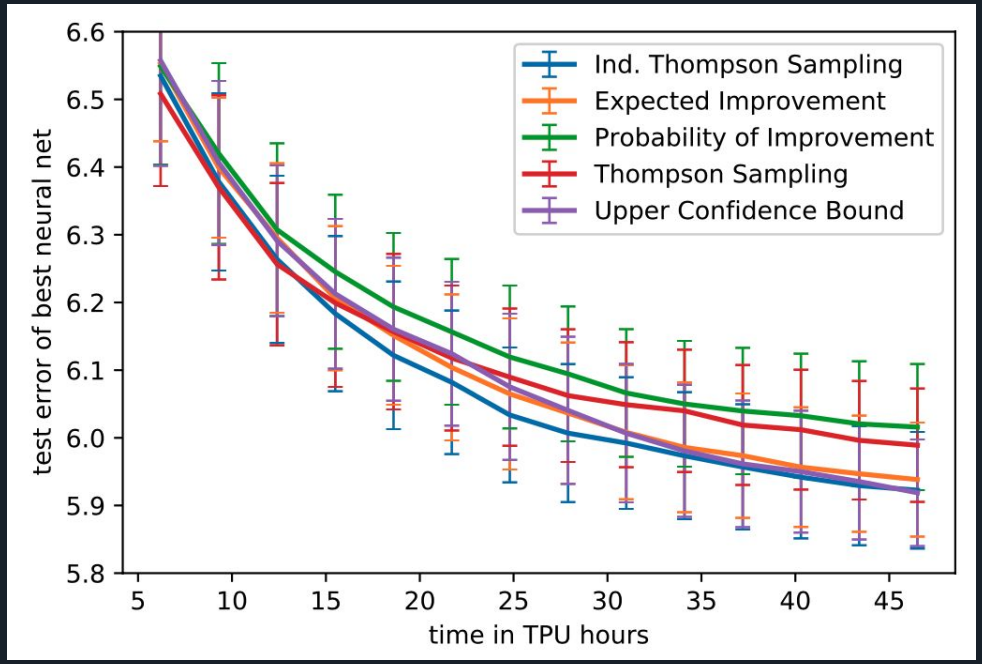
Standalone (MAE) | Uncertainty (RMSCE) | Perf. in BO framework

GraphNN and path-encoding perform best

Acquisition Function

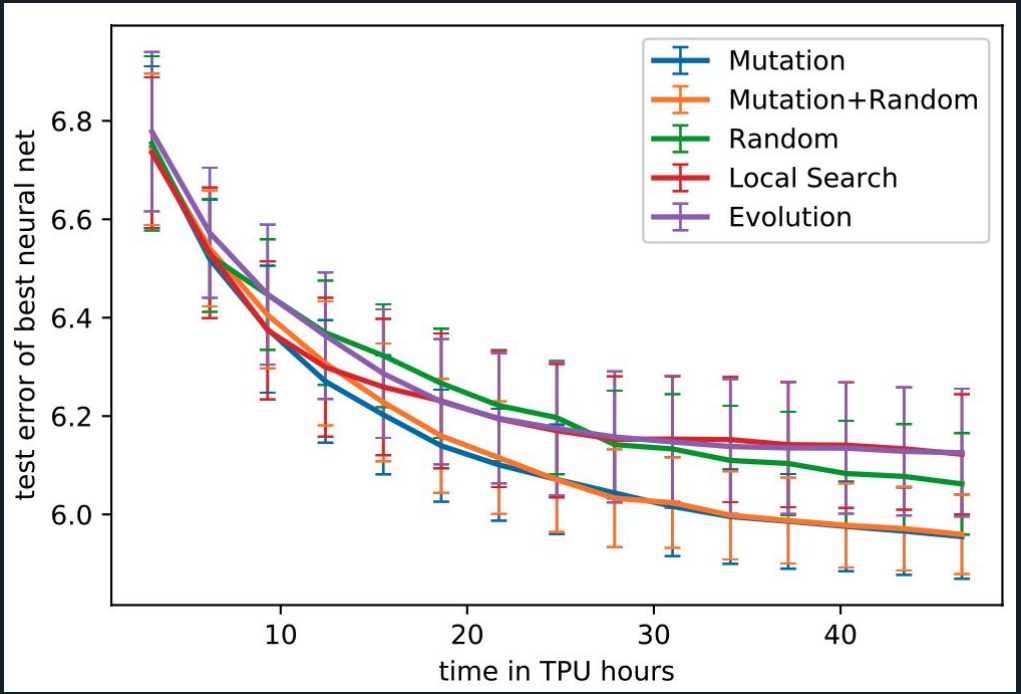
- Exploration vs. exploitation





Acquisition Optimization

- *Small mutations* of the best architectures is best
- Predictions are most accurate when close to training data



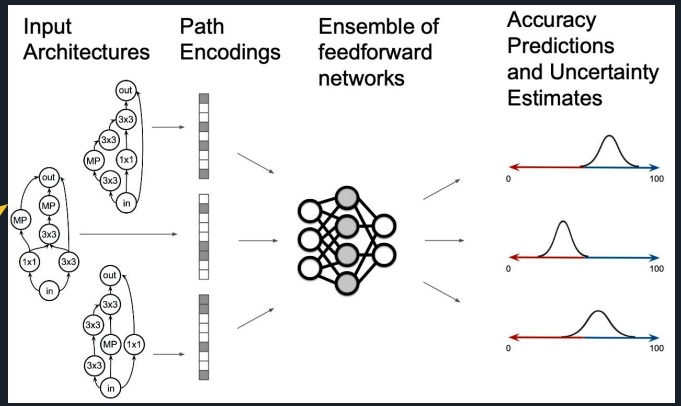
BANANAS

Algorithm 1 BANANAS

Input: Search space A, dataset D, parameters t_0, T, M, c, x, acquisition function ϕ, function $f(a)$ returning validation error of a after training.

1. Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from A and train them on D.
2. For t from t_0 to T,
   i. Train an ensemble of meta neural networks on $\{(a_0, f(a_0)), \ldots, (a_t, f(a_t))\}$.
   ii. Generate a set of c candidate architectures from A by randomly mutating the x architectures a from $\{a_0, \ldots, a_t\}$ that have the lowest value of $f(a)$.
   iii. For each candidate architecture a, evaluate the acquisition function $\phi(a)$.
   iv. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$, and evaluate $f(a_{t+1})$.

Output: $a^* = \text{argmin}_{t=0,\ldots,T} f(a_t)$.

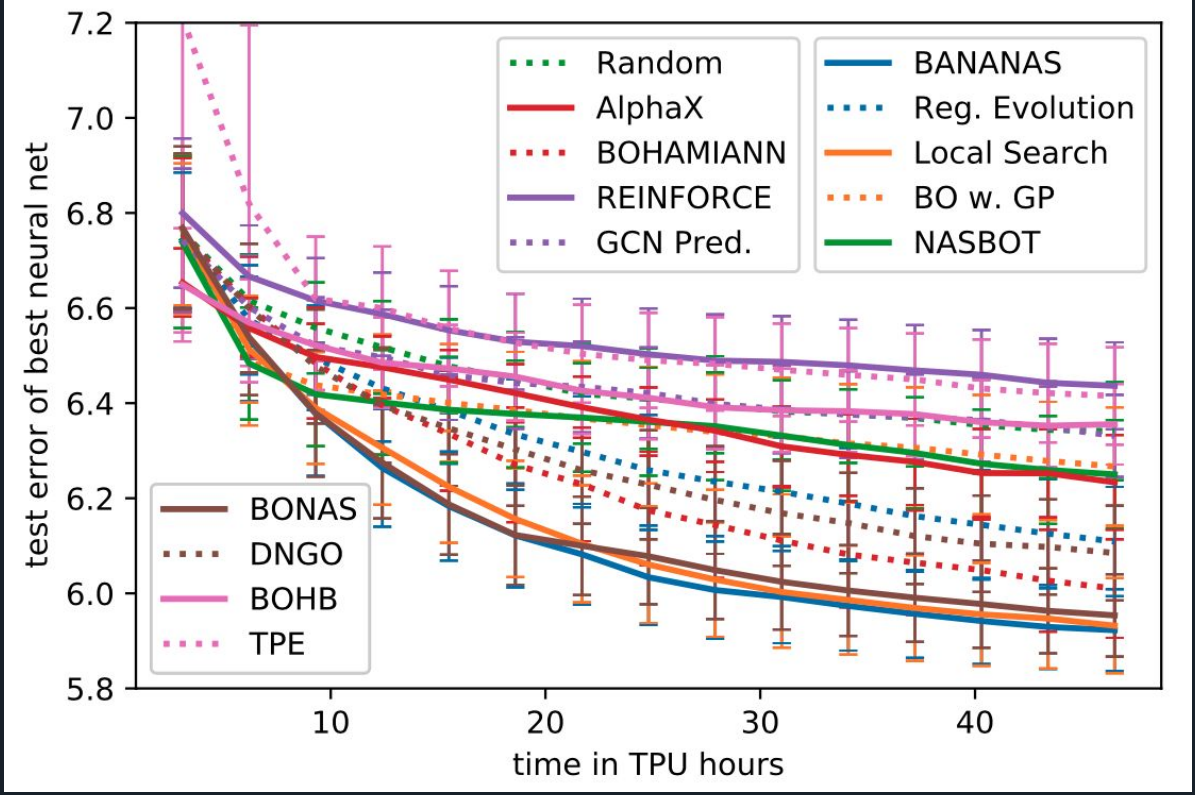


Path encoding, ensemble

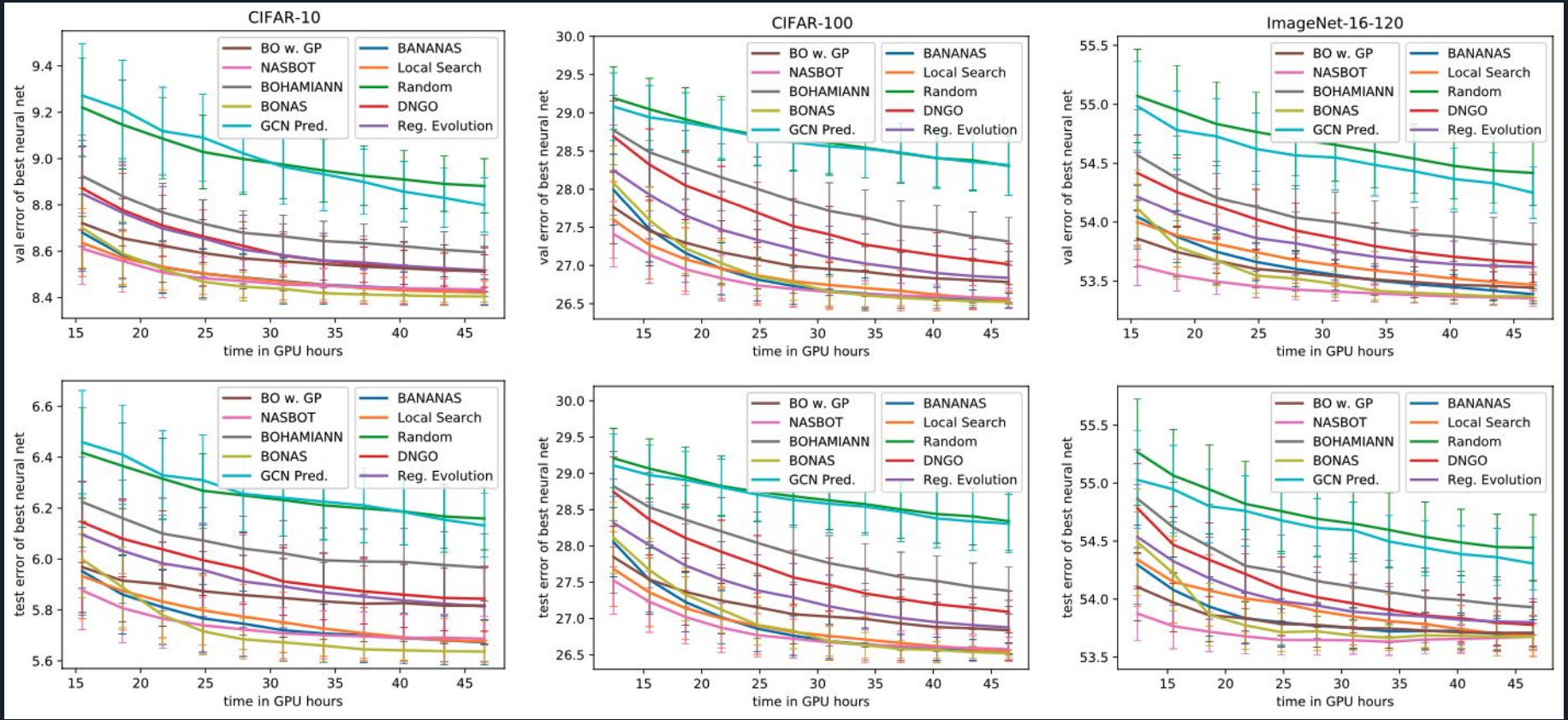
Small mutations

Independent Thompson Sampling

NASBench-101



NASBench-201

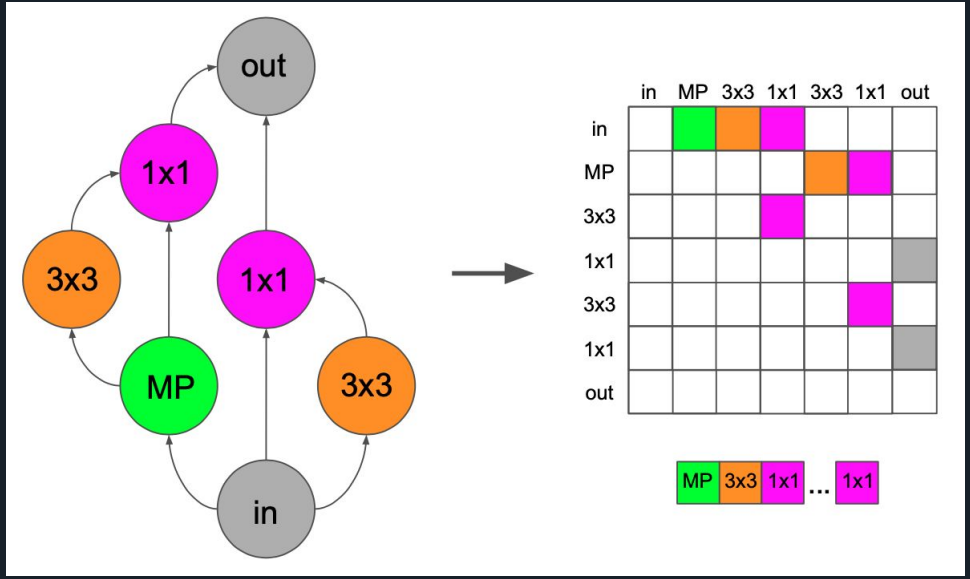


Outline

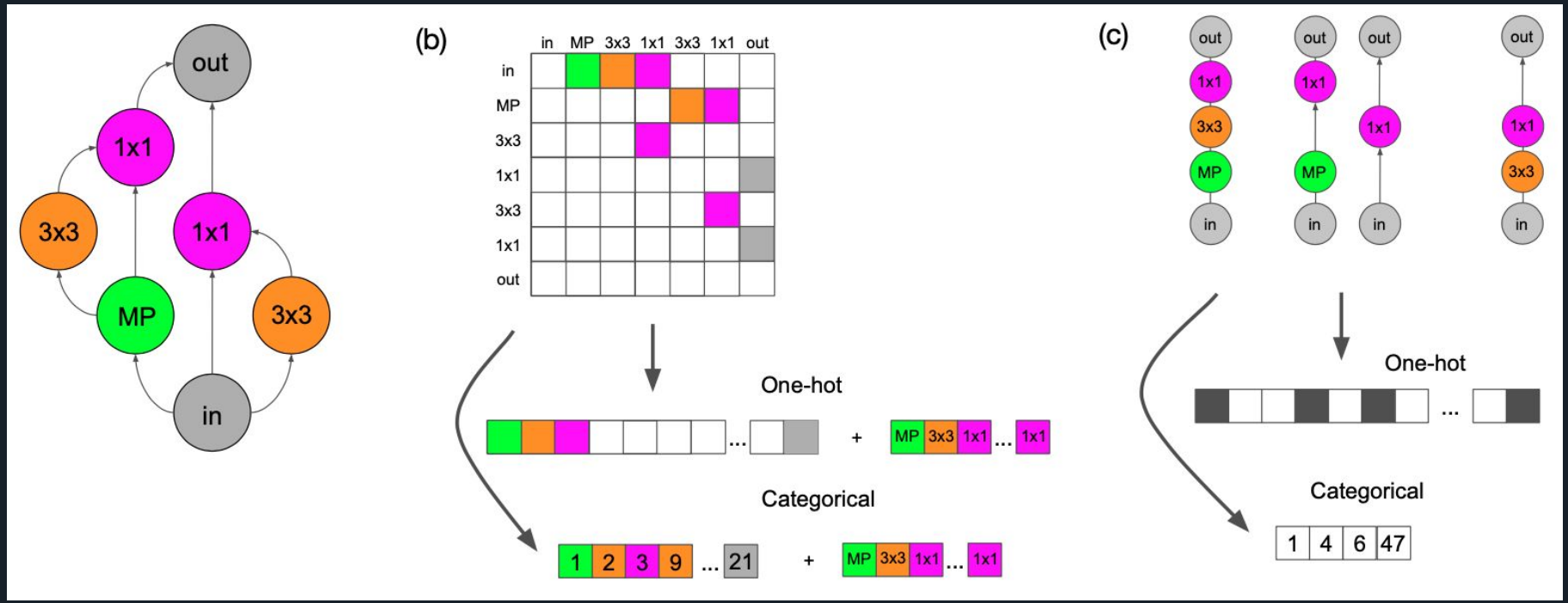
- Introduction to AutoML
- Performance Prediction
- Neural Architecture Search
- Architecture Encodings
- Conclusion

Adjacency Matrix Encoding

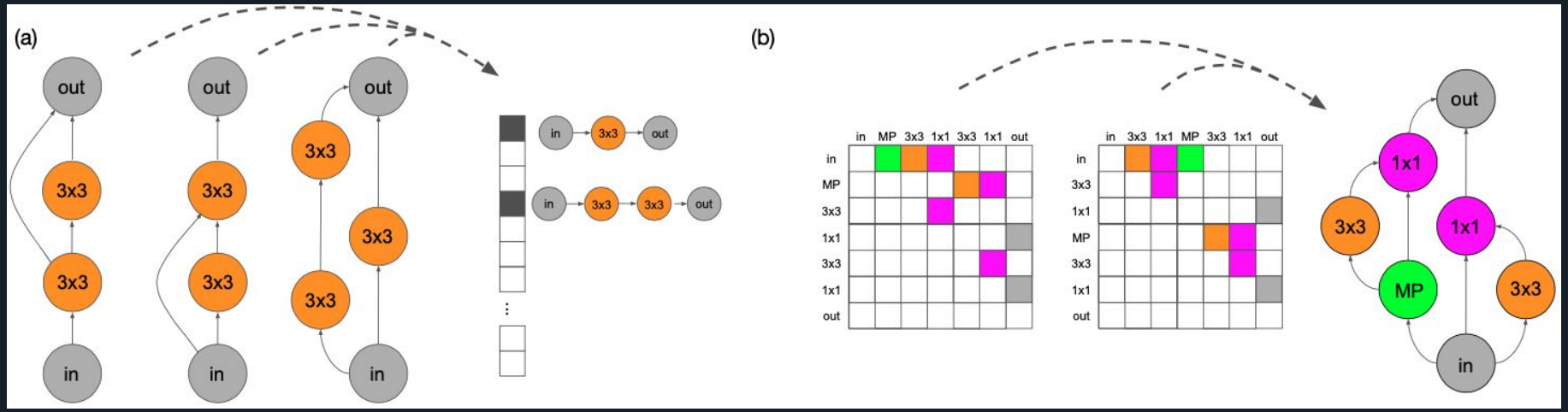
Most NAS algorithms search over DAG-based architectures, which must be encoded into a tensor



Encodings for NAS



Not one-to-one or well-defined



Path-based encoding:
not one-to-one

Adjacency-based encoding:
not well-defined

NAS encoding-dependent subroutines

Many NAS algos can be composed from three subroutines

- Sample random architecture
- Perturb architecture
- Train predictor model

Algorithm 1 BANANAS

Input: Search space A, dataset D, parameters t_0, T, M, c, x, acquisition function ϕ, function $f(a)$ returning validation error of a after training.

1. Draw t_0 architectures $a_0, \ldots, a_{t_0}$ uniformly at random from A and train them on D.
2. For t from t_0 to T,
   i. Train an ensemble of neural predictors on $\{(a_0, f(a_0)), \ldots, (a_t, f(a_t))\}$ using the path encoding to represent each architecture.
   ii. Generate a set of c candidate architectures from A by randomly mutating the x architectures a from $\{a_0, \ldots, a_t\}$ that have the lowest value of $f(a)$.
   iii. For each candidate architecture a, evaluate the acquisition function $\phi(a)$.
   iv. Denote a_{t+1} as the candidate architecture with minimum $\phi(a)$, and evaluate $f(a_{t+1})$.

Output: $a^* = \text{argmin}_{t=0,\ldots,T} f(a_t)$.

Algorithm 1 Aging Evolution

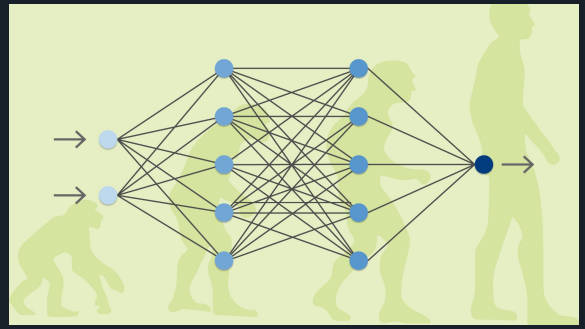
```
population ← empty queue                          ▷ The population.
history ← ∅                                       ▷ Will contain all models.
while |population| < P do                         ▷ Initialize population.
    model.arch ← RANDOMARCHITECTURE()
    model.accuracy ← TRAINANDEVAL(model.arch)
    add model to right of population
    add model to history
end while
while |history| < C do                            ▷ Evolve for C cycles.
    sample ← ∅                                    ▷ Parent candidates.
    while |sample| < S do
        candidate ← random element from population
                ▷ The element stays in the population.
        add candidate to sample
    end while
    parent ← highest-accuracy model in sample
    child.arch ← MUTATE(parent.arch)
    child.accuracy ← TRAINANDEVAL(child.arch)
    add child to right of population
    add child to history
    remove dead from left of population           ▷ Oldest.
    discard dead
end while
return highest-accuracy model in history
```

Algorithm 1 Bayesian Optimized Neural Architecture Search (BONAS). $\mathcal{A}$ is the given search space, N is the number of initial architectures, k is the ratio of GCN / BLR update times.

1: initialize random N fully-trained architectures: $\mathcal{D} = \{(A_i, X_i, t_i)\}_{i=1}^{N}$ from search space $\mathcal{A}$;
2: initial training of GCN using $\mathcal{D}$ with proposed loss;
3: replace the final layer of GCN with BLR;
4: initialize Sampler;
5: **repeat**
6: **for** iteration$= 1, 2, \ldots, k$ **do**
7: sample candidate pool $\mathcal{C}$ from $\mathcal{A}$;
8: **for** each candidate m in $\mathcal{C}$ **do**
9: embed m using GCN;
10: compute μ and σ^2 in (4) and (5) using BLR;
11: compute expected improvement (EI) in (2);
12: **end for**
13: $M \leftarrow$ candidate with the highest EI score;
14: fully train M to obtain its actual performance;
15: add M and its actual performance to $\mathcal{D}$;
16: update BLR using the enlarged $\mathcal{D}$;
17: update Sampler;
18: **end for**
19: retrain GCN using the enlarged $\mathcal{D}$ with proposed loss;
20: **until** stop criterion satisfy.

Perturb/Mutate Architecture

Mutate a path vs. mutate an edge/operation



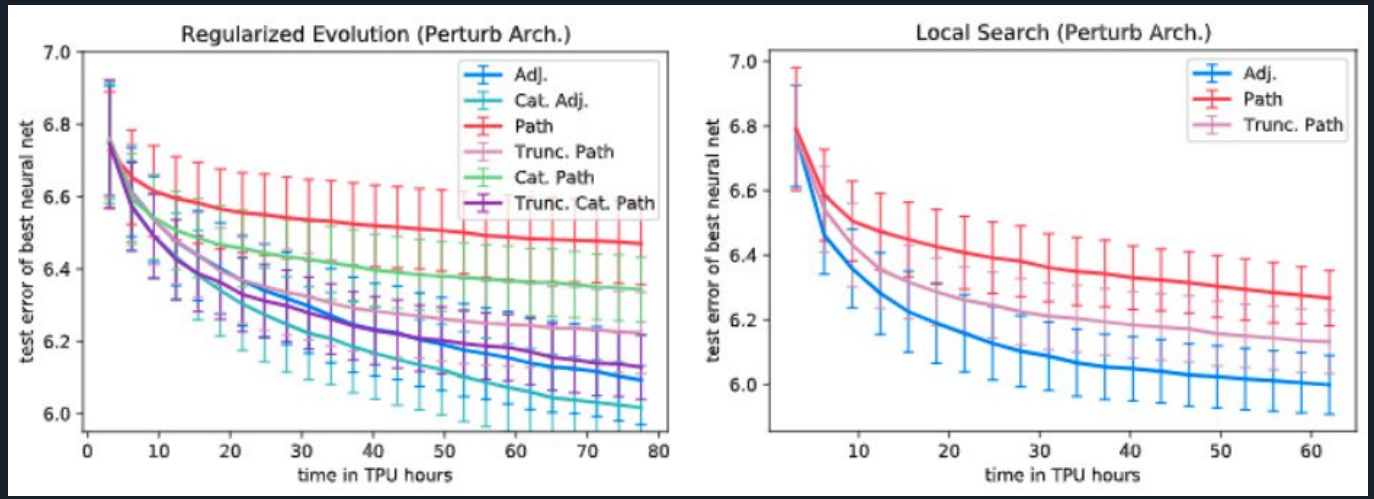
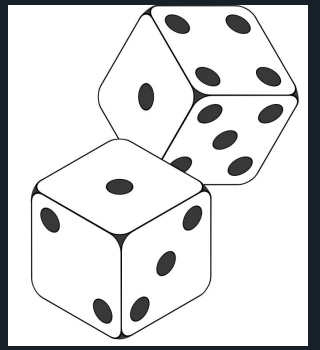
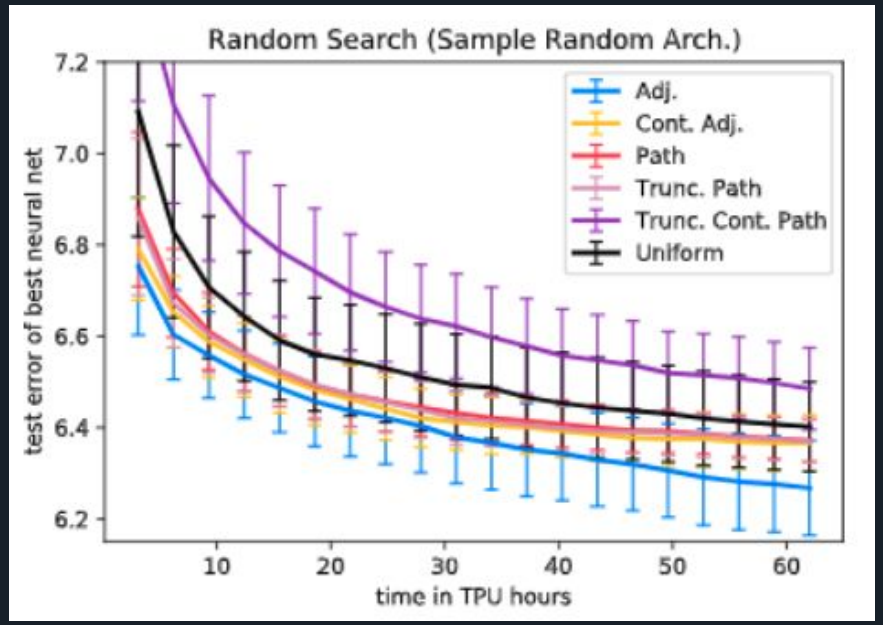


Image source: https://www.inovex.de/blog/neuroevolution/

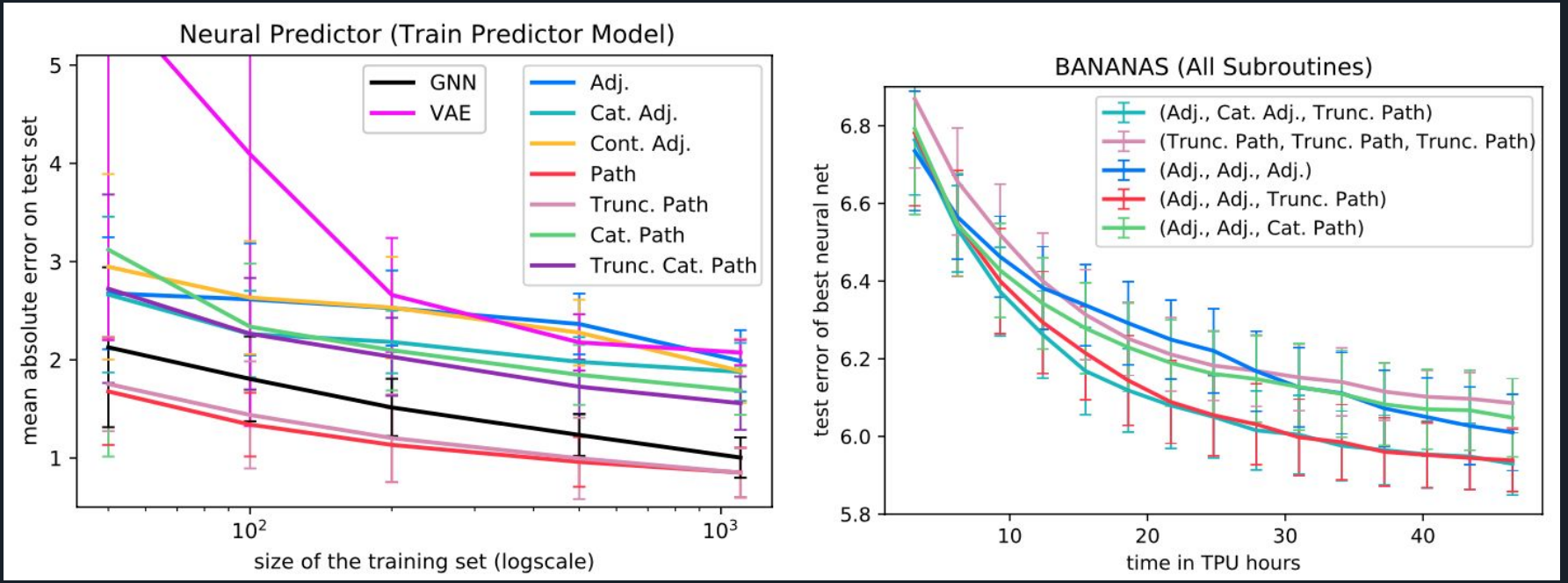
Sample Random Architecture

Each encoding gives a different distribution that we can sample from

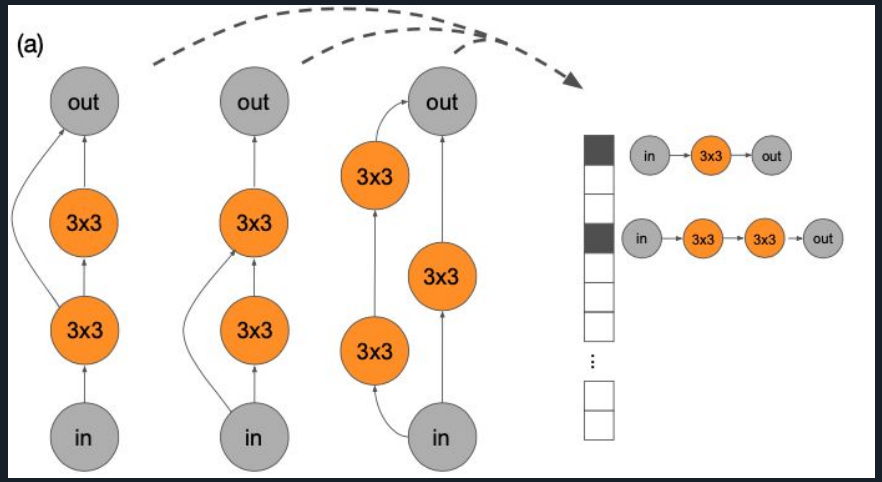




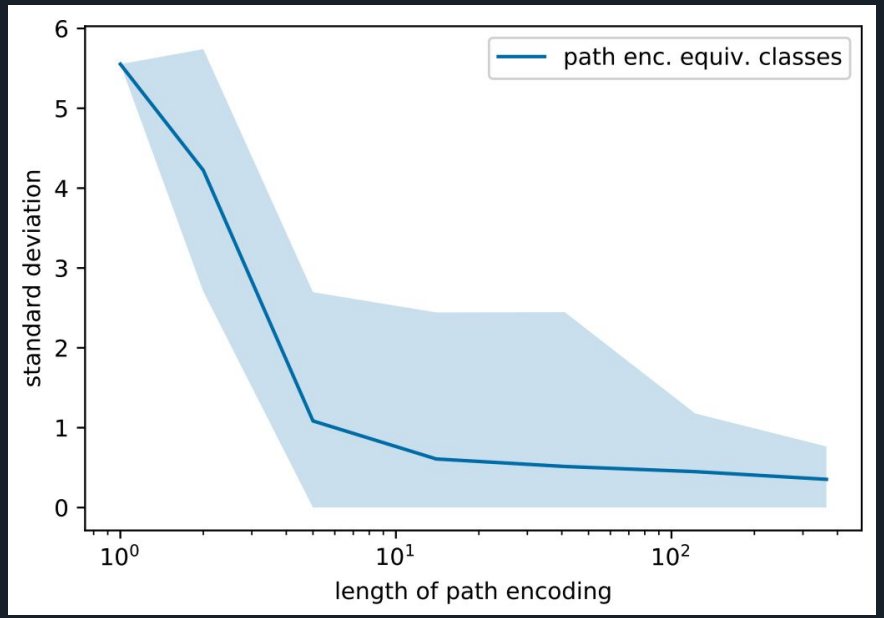
Train Predictor Model



Equivalence classes



Non one-to-one is a *benefit*



Conclusions

- Large-scale study of performance predictors
  - OMNI achieves the best performance
- “BO + Neural Predictor” is a powerful NAS framework
  - BANANAS
- Encodings are an important design decision

https://abacus.ai/publications

Code: https://github.com/automl/naslib

https://github.com/naszilla/naszilla

Thanks! Questions?