NAS-Bench-x11
and the Power of Learning Curves

Shen Yan
Michigan State University

Colin White
Abacus.AI

Yash Savani
Carnegie Mellon University

Frank Hutter
University of Freiburg
Bosch Center for AI
One-slide summary:

- We give a new technique to create surrogate NAS benchmarks with realistic learning curves.
- We create NAS-Bench-111, NAS-Bench-311, and NAS-Bench-NLP11.
- We use these to show popular NAS algorithms can be further improved by adding learning curve extrapolation.
Neural architecture search

- Notoriously challenging to give fair comparisons [Li & Talkwalkar 2019], [Hutter & Lindauer 2020]
  - Computationally intensive
  - No common search spaces
Tabular NAS Benchmarks

Train all architectures in a search space

Used to simulate NAS experiments

- NAS-Bench-101 [Ying et al. 2019]
  - Size 423k
- NAS-Bench-201 [Dong & Yang 2019]
  - Size 15k

```python
# Load the data from file (this will take some time)
nasbench = api.NASBench('/path/to/nasbench.tfrecord')

# Create an Inception-like module (5x5 convolution replaced with two 3x3
# convolutions).
model_spec = api.ModelsSpec(
    # Adjacency matrix of the module
    matrix=[[0, 1, 1, 1, 0, 1, 0], 
            [0, 0, 0, 0, 0, 0, 1], 
            [0, 0, 0, 0, 0, 0, 1], 
            [0, 0, 0, 0, 0, 0, 1], 
            [0, 0, 0, 0, 0, 0, 1]], 
    # input layer
    0, 0, 0, 0, 0, 0, 1], 
    # 3x3 conv
    0, 0, 0, 0, 0, 0, 1], 
    # 3x3 conv
    [0, 0, 0, 0, 0, 0, 1], 
    # 5x5 conv (replaced by two 3x3's)
    [0, 0, 0, 0, 0, 0, 1], 
    # 5x5 conv (replaced by two 3x3's)
    [0, 0, 0, 0, 0, 0, 1], 
    # 3x3 max-pool
    [0, 0, 0, 0, 0, 0, 1]], 
    # output layer
    # Operations at the vertices of the module, matches order of matrix
    ops=[INPUT, CONV1X1, CONV3X3, CONV3X3, CONV3X3, MAXPOOL3X3, OUTPUT])

# Query this model from dataset, returns a dictionary containing the metrics
# associated with this model.
data = nasbench.query(model_spec)
```

NAS-Bench-101
Surrogate NAS Benchmarks

- **NAS-Bench-301** [Siems et al. 2020]
  - Based on DARTS search space
  - Size $10^{18}$

Enables much larger NAS Benchmarks

<table>
<thead>
<tr>
<th>NAS methods</th>
<th># eval</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS (Bergstra &amp; Bengio, 2012)</td>
<td>23746</td>
</tr>
<tr>
<td>Evolution</td>
<td></td>
</tr>
<tr>
<td>DE (Awad et al., 2020)</td>
<td>7275</td>
</tr>
<tr>
<td>RE (Real et al., 2019)</td>
<td>4639</td>
</tr>
<tr>
<td>BO</td>
<td></td>
</tr>
<tr>
<td>TPE (Bergstra et al., 2011)</td>
<td>6741</td>
</tr>
<tr>
<td>BANANAS (White et al., 2019)</td>
<td>2243</td>
</tr>
<tr>
<td>COMBO (Chen et al., 2019)</td>
<td>745</td>
</tr>
<tr>
<td>One-Shot</td>
<td></td>
</tr>
<tr>
<td>DARTS (Liu et al., 2019b)</td>
<td>2053</td>
</tr>
<tr>
<td>PC-DARTS (Xu et al., 2020)</td>
<td>1588</td>
</tr>
<tr>
<td>DrNAS (Chen et al., 2020)</td>
<td>947</td>
</tr>
<tr>
<td>GDAS (Dong &amp; Yang, 2019)</td>
<td>234</td>
</tr>
</tbody>
</table>

Table 2: NAS methods used to cover the search space.
NAS Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Size</th>
<th>Queryable</th>
<th>Based on</th>
<th>Full train info</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAS-Bench-101</td>
<td>423k</td>
<td>✓</td>
<td></td>
<td>✗</td>
</tr>
<tr>
<td>NAS-Bench-201</td>
<td>6k</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>NAS-Bench-NLP</td>
<td>$10^{53}$</td>
<td>✗</td>
<td></td>
<td>✗</td>
</tr>
<tr>
<td>NAS-Bench-301</td>
<td>$10^{18}$</td>
<td>✓</td>
<td>DARTS</td>
<td>✗</td>
</tr>
<tr>
<td>NAS-Bench-ASR</td>
<td>8k</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

No learning curves - can only simulate black-box algorithms!
## NAS Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Size</th>
<th>Queryable</th>
<th>Based on</th>
<th>Full train info</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAS-Bench-101</td>
<td>423k</td>
<td>✓</td>
<td></td>
<td>❌</td>
</tr>
<tr>
<td>NAS-Bench-201</td>
<td>6k</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>NAS-Bench-NLP</td>
<td>$10^{53}$</td>
<td>✗</td>
<td></td>
<td>❌</td>
</tr>
<tr>
<td>NAS-Bench-301</td>
<td>$10^{18}$</td>
<td>✓</td>
<td>DARTS</td>
<td>❌</td>
</tr>
<tr>
<td>NAS-Bench-ASR</td>
<td>8k</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>NAS-Bench-111</td>
<td>423k</td>
<td>✓</td>
<td>NAS-Bench-101</td>
<td>✓</td>
</tr>
<tr>
<td>NAS-Bench-311</td>
<td>$10^{18}$</td>
<td>✓</td>
<td>DARTS</td>
<td>✓</td>
</tr>
<tr>
<td>NAS-Bench-NLP11</td>
<td>$10^{53}$</td>
<td>✓</td>
<td>NAS-Bench-NLP</td>
<td>✓</td>
</tr>
</tbody>
</table>

No learning curves - can only simulate black-box algorithms!
Roadmap

- Motivation
- **Generating Learning Curves**
- Evaluation
- The Power of Learning Curves
- Conclusion
Generating Learning Curves

We can’t just use a surrogate model to predict the entire learning curve.

Generating realistic noise is critical.

This would lead to de-noised learning curves.

Figure 1: Number of architectures used for training the GIN surrogate model vs MAE on the NAS-Bench-101 dataset.
Generating Learning Curves

Generating **realistic noise** is critical

**Goal:** given architecture encoding, predict a **distribution**
Two-part technique

(1) Predict mean LC

(2) Predict noise
Predicting the mean learning curve

SVD helps to reduce the noise

Compress the learning curves from the training set

Predict only the top 5 principal components
Predicting the mean learning curves
Noise modeling

Compute the residuals, then use a sliding window to approximate STDev’s
Full technique
NAS-Bench-x11

We create

- NAS-Bench-111
  - Created a new training set of size 1500
- NAS-Bench-311
  - Used the 60k architectures from NAS-Bench-301
- NAS-Bench-NLP11
  - Used the 14k architectures from NAS-Bench-NLP
  - Improved by adding acc’s from first three epochs

API and surrogate benchmarks: [https://github.com/automl/NAS-Bench-x11](https://github.com/automl/NAS-Bench-x11)
Evaluation (mean learning curves)

<table>
<thead>
<tr>
<th></th>
<th>Avg. $R^2$</th>
<th>Final $R^2$</th>
<th>Avg. KT</th>
<th>Final KT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabular (1 seed)</td>
<td>0.553</td>
<td>0.778</td>
<td>0.529</td>
<td>0.654</td>
</tr>
<tr>
<td>Tabular (2 seeds)</td>
<td>0.672</td>
<td>0.845</td>
<td>0.581</td>
<td>0.709</td>
</tr>
<tr>
<td>Tabular (3 seeds)</td>
<td>0.707</td>
<td>0.854</td>
<td>0.602</td>
<td>0.718</td>
</tr>
<tr>
<td>Tabular (4 seeds)</td>
<td>0.727</td>
<td>0.870</td>
<td>0.617</td>
<td>0.732</td>
</tr>
<tr>
<td>NAS-Bench-311</td>
<td>0.715</td>
<td>0.838</td>
<td>0.628</td>
<td>0.711</td>
</tr>
</tbody>
</table>

Similar rank correlation to a 3-seed tabular benchmark
**Evaluation (noise model)**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Avg. $R^2$</th>
<th>Final $R^2$</th>
<th>Avg. KT</th>
<th>Final KT</th>
<th>Avg. KL</th>
<th>Final KL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAS-Bench-111</td>
<td>0.529</td>
<td>0.630</td>
<td>0.531</td>
<td>0.645</td>
<td>2.016</td>
<td>1.061</td>
</tr>
<tr>
<td>NAS-Bench-111 (w. accs)</td>
<td>0.630</td>
<td>0.853</td>
<td>0.611</td>
<td>0.794</td>
<td>1.710</td>
<td>0.926</td>
</tr>
<tr>
<td>NAS-Bench-311</td>
<td>0.779</td>
<td>0.800</td>
<td>0.728</td>
<td>0.788</td>
<td>0.905</td>
<td>0.600</td>
</tr>
<tr>
<td>NAS-Bench-NLP11</td>
<td>0.326</td>
<td>0.314</td>
<td>0.505</td>
<td>0.475</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NAS-Bench-NLP11 (w. accs)</td>
<td>0.878</td>
<td>0.895</td>
<td>0.878</td>
<td>0.844</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Spike anomalies**

Compare probability of anomalies of surrogates vs. real data
Roadmap

- Motivation
- Generating Learning Curves
- Evaluation
- The Power of Learning Curves
- Conclusion
Learning Curve Extrapolation (LCE)

Used to speed up black-box NAS algorithms

- Reg. Evolution, BANANAS, local search, etc

Use LCE to stop training bad architectures early

[Domhan et al. 2015], [Baker et al. 2017]
Algorithm 1 Single-Fidelity Algorithm

1: initialize history
2: while $t < t_{\text{max}}$:
3:    arches = gen_candidates(history)
4:    accs = train(arches, epoch=$E_{\text{max}}$)
5:    history.update(arches, accs)
6:    Return arch with the highest acc

Algorithm 2 LCE Framework

1: initialize history
2: while $t < t_{\text{max}}$:
3:    arches = gen_candidates(history)
4:    accs = train(arches, epoch=$E_{\text{final}}$)
5:    sorted_by_pred = LCE(arches, accs)
6:    arches = sorted_by_pred[0:n]
7:    accs = train(arches, epoch=$E_{\text{max}}$)
8:    history.update(arches, accs)
9:    Return arch with the highest acc
Conclusions & Future Work

- New technique: surrogate benchmarks with full training information
  - Learning curves with realistic noise
- NAS-Bench-111, NAS-Bench-311, NAS-Bench-NLP11
- Framework to add LCE to black-box NAS algorithms

Code: [https://github.com/automl/NAS-Bench-x11](https://github.com/automl/NAS-Bench-x11)

Thanks!