Automated Machine Learning: State-of-The-Art and Open Challenges

Radwa Elshawi Mohamed Maher Sherif Sakr
Problem definition

- CASH Problem: Combined Algorithm Selection and Hyper-parameter tuning

\[ A^{(i)*} \in \arg\min_{A \in A} L(A^{(i)}, D_{\text{train}}, D_{\text{validation}}) \]

- Goal: Select an ML algorithm that can achieve optimal performance in both the training and validation set, given a specific time budget constraint.
Meta-Learning

- Warm starting of the optimization algorithm by leveraging previous learning experiences.
- **Learning based on task properties:**
  - Use constructed meta-features to compute the similarity between two datasets
  - Build a meta-model that learns the statistical properties of the previous dataset
- **Learning from previous model evaluations:**
  - Train a meta-learner on previous evaluations of the ML model (parameters $\theta_i$, task $t_i$) to predict new parameters $\Theta^*_{\text{new}}$ for a new task $t^*$
- **Learning from already pretrained models:**
  - Transfer Learning: utilize pretrained models, trained on a task $t_i$, for training on a new task $t_j$ given certain similarity between $t_i$ and $t_j$
Neural Architecture Search for Deep Learning

- Technique for automatic design of deep Artificial Neural Networks
- **Random Search**: Randomly sample neural architectures from a bounded search space
- **Reinforcement Learning**: An agent explores the finite search space of architectures with the goal of maximizing the performance on the given task
- **Gradient-based optimization**: Explore the continuous search space with the help of gradient descent
- **Evolutionary Methods**: They are based on genetic algorithms or hierarchical evolution
- **Bayesian optimization**: They rely on Gaussian processes or tree-based models
Hyperparameter Optimization (1)

- **Black-Box optimization:** The analytical form of the objective function is not known, only the results of evaluating this function at different points can be used.
- **Grid Search \ Random Search:**
  - Easily parallelizable methods
  - Given a computational budget, random search is better
- **Bayesian Optimization:** Suitable for computationally expensive objective functions
- **Simulated Annealing:** Update each hyperparameter value based on the neighbourhood states
- **Genetic Algorithms:** Apply genetic operations to a population of hyperparameter configurations
Hyperparameter Optimization (2)

- **Multi-fidelity optimization**: Use a small part of the dataset to evaluate the objective function (low-fidelity evaluation)
  - Use many low-fidelity evaluation instead of one high-fidelity evaluation
  - Trade-off between computational cost and optimization performance
- **Modeling learning curves**: Uses the progression of the learning curve as the stopping criterion for the hyperparameter optimization
- **Successive halving**: multi-fidelity method which keeps only the best half of the tested configurations at each step
- **HyperBand**: multi-fidelity method that applies Successive halving on randomly sampled configurations
Tools and Frameworks

- **Centralized Frameworks:**
  - designed to run on a single machine
  - suitable for handling small to medium datasets

- **Distributed Frameworks:**
  - Utilize multi-node systems for solving the CASH problem for bigger datasets

- **Cloud-based Frameworks:**
  - Utilize the computational power of cloud-based environments
  - Typically require minimal user experience
  - Compatible with other services in the same cloud ecosystem

- **Neural Network Automation Frameworks:**
  - find neural network architectures that are competitive with architectures designed by human expert
Other automation aspects

- Pre-Modeling
  - Data Understanding
    - Sanity Checking
    - Feature Based Analysis
    - Data Lifecycle Analysis
  - Data Validation
    - Automatic Correction
    - Automatic Alerting
  - Data Preparation
    - Feature Synthesis
    - Feature Addition

- Post-Modeling
  - Model Tracking
  - Model Deployment

- Further Automation Tasks
Summary

● Good overview of all available methods and tools for automatizing the ML pipeline
● It would be probably insightful to analyse all the presented methods from the perspective of the budget constraint
● Expand the survey to include AutoML methods for massive neural architectures
Hyperparameter Optimization
• Machine learning algorithms can be very sensitive to hyperparameter settings. So choosing the right one is crucial.

• Hyperparameters
  ○ Learning rate
  ○ Regularization
  ○ Architecture

• Budget
BOHB: Robust and Efficient Hyperparameter optimization at Scale

Stefan Falkner, Aaron Klein, Frank Hutter

PMLR 2018
Motivation

- Bayesian based methods for HPO are typically computationally infeasible.
- Random search based methods (such as Hyperband) are faster but do not converge to good solutions.

Contribution

- Create a best of both worlds approach called BOHB
Bayesian Optimization (BO)

- Density over input configuration space is estimated to select new candidate configurations to evaluate.

Hyperband (HB)

- Cheap to evaluate approx. versions of objective function are defined given a budget. Higher the budget higher the quality of the estimate.
- Budget here can be the num. of iterations, num. of data points, num. of steps in MCMC algo., num. of trials in deep reinforcement learning etc.
- Invoke successive halving (SHA) to select and promote configurations.
Successive Halving (SHA)

- Distribute the budget over ‘n’ random initial configurations.
- Evaluate and promote the best half to the next rung with double the budget.
- Iterate till one remains.
BOHB

Algorithm 2: Pseudocode for sampling in BOHB

| input  | observations $D$, fraction of random runs $\rho$, percentile $q$, number of samples $N_s$, minimum number of points $N_{min}$ to build a model, and bandwidth factor $b_w$ |
|------------------|
| output | next configuration to evaluate |

1. if $\text{rand}() < \rho$ then return random configuration
2. $b = \arg \max \{D_b : |D_b| \geq N_{min} + 2\}$
3. if $b = \emptyset$ then return random configuration
4. fit KDEs according to Eqs. (2) and (3)
5. draw $N_s$ samples according to $l'(x)$ (see text)
6. return sample with highest ratio $l(x)/g(x)$

- BO does model based search for suitable configurations.
- HB selects the num. of configurations and assigns budget.
- Model is updated based on evaluated configurations.
- Iterate.
Parallel resources

- In BOHB multiple configurations need to be evaluated independently at each iteration.
- This can be parallelized.
Evaluation over various scenarios
Best of both worlds: BOHB

- Optimizing 6 hyperparameters of a neural network.
- Has strong anytime performance obtained from HB.
- Has strong final performance obtained from BO.
Stochastic Counting Ones

8 + 8 dimensions

normalized regret

10^0

10^1

10^2

10^3

10^4

cumulative budget / b_{m,a,x}

RS  HB  TPE  BOHB  SMAC
Support Vector Machines

- Setup: Surrogate imitates optimization of SVM with RBF kernel
- Tunable hyperparameters (2)
- Budget: num. of training data points for HB, BOHB, full set for the others.
- Take away: GP-BO and RS are too slow. Fabolas, HB and BOHB find good configuration quickly with Fabolas having the fastest initial speedup.
Feed-forward Neural Networks

![Graph showing regret vs. wall clock time for different methods in poker](image-url)
Bayesian Neural Networks

- Setup: 2 layered fully connected Bayesian neural network trained with MCMC sampling.
- Tunable hyperparameters (4):
  - step length,
  - length of burn-in period,
  - num. of units in each layer,
  - decay parameter of momentum.
- Dataset: Boston housing
- Budget: 500-10000 MCMC steps

Take away: BOHB converged faster than both HB and TPE (BO) and even found a better configuration than the baselines.
Reinforcement Learning

- Setup: Proximal policy optimization to learn the cartpole swing-up task.
- Tunable hyperparameters (8)
- Budget: BOHB and HB 1-9 trials, others fixed 9 trials.
- Take away: BOHB starts same as HB but converged to better configurations. TPE did not have enough budget to find the same.
Convolutional Neural Networks

- Tunable hyperparameters: learning rate, momentum, weight decay, batch size.
- Budget: 22,66,200,600 epochs.
- 19 parallel workers
- Take away: BOHB is practically useful for resource constrained optimization.
Limitations

- Small budgets gives us cheap approximations of objective function.
- This assumes that relative ranking of configurations mostly hold even for small budgets.
- If this is not true and the approximation is too noisy then it will result in BHOB being slower than BO and worse than even random search.
- To overcome this BHOB samples a fixed fraction (1/3) of configurations randomly. This avoids missing good configurations hiding among bad ones.
Summary

- BOHB combines good initial performance of HB and good convergence properties of BO.
- BO component helps guide the search and results in faster convergence.
- HB component helps get a quick start through SHA and results in good initial speedup.
- Solution is robust, flexible, scalable gives strong anytime and final performance.
- Code has been made available.
A System for Massively Parallel Hyperparameter Tuning

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, Ameet Talwalkar

MLSys 2020
Motivation

- Usually, with increasingly large models, the available additional budget for HPO is low.
- Massive parallelization is the way to go.
- Adaptive search based methods are iterative and hard to parallelize.
- Grid and random search are trivial to parallelize but don’t scale well with increased num. of hyperparameters.
- Synchronous hyperparameter tuning methods struggle from stragglers.
- There is a need for production grade hyperparameter tuning systems.
Contribution

- Asynchronous SHA called ASHA.
- A method to parallelize ASHA.
- Deploy ASHA in a production grade system.
Async SHA: ASHA

- When a worker finishes a job it requests a new one.
- We look at the rungs from top to bottom to see if there are configurations in the top 1/\( \eta \) of each rung.
- These are promoted to the next rung.
- If none, we assign the worker to add a configuration to the lowest rung to grow the width of the level so that more configurations can be promoted.
Single Machine Experiments

- PBT is the state-of-the-art evolutionary method that iteratively improves fitness of configurations.
- Benchmark 1:
  - SHA and ASHA do better than PBT.
- Benchmark 2:
  - SHA, ASHA and PBT do comparably.
- Both cases, ASHA does not degrade performance even though it is async.
Distributed experiments

- 25 workers.
- **Benchmark 1:**
  - ASHA better than PBT, comparable to BOHB.
  - ASHA took 40 min to evaluate 1000 configurations and find a good one using 25 workers.
  - This is about the same time as needed to train for a single configuration on 1 worker.
  - In a serial setting this would take 400 min.
  - Increase in # workers by 25 led to only 10 X speedup due to relative simplicity of the task.
- **Benchmark 2:**
  - ASHA comparable to PBT, better than BOHB.
  - ASHA took 25 min while using 25 workers.
  - It took 700 min while using 1 worker.
  - Speedup in linear since this is a harder task that can leverage the additional workers.
Distributed Large-scale experiments

- 500 workers.
- Vizier is Google’s internal hyperparameter optimizer.
Production grade ASHA

- Simplified user interface with same input as random search.
- Stopping criteria is a fixed number of configurations.
- Use Paleo predicted trade-off curves to choose number of GPUs per configuration for a given efficiency.
- A fair share scheduler that adaptively allocated resources over the lifetime of a job.
- Reproducible checkpoints and state saving for pause and restart feature.
Summary

- Making SHA async (ASHA) does not degrade its performance and allows for massive parallelization of HPO.
- Parallelized ASHA does comparable or better than PBT, SHA and BOHB.
- A production grade HPO system was developed with many contributions in the form of informed design decisions.
Network Architecture Search (NAS)
Motivation

Main problem: large discrete search space

SOTA methods are gradient-free:
- Black-box search in a discrete non-differentiable space
- Significantly slower

[Elshawi et al., 2019]
Main contribution

A method for gradient-based search for network architecture:

- Main idea: relax the discrete set of candidate architecture to be a continuous space and apply gradient descent
- Orders of magnitude gain in computation time due to the better efficiency of gradient-based optimization
- Generic enough for CNN and RNN
NASNet Search Space and the Cell [Zoph et al., 2018]

Steps:
1. Constrain all cells to have the same architecture
2. Design a small outer structure and search for a cell architecture on a smaller dataset (CIFAR-10)
3. Preserve the cell architecture and scale up to a larger outer structure
4. Train on a larger dataset (ImageNet) that we really want

Fix outer structure ⇒ search operations in a cell ⇒ scale up outer structure ⇒ train CNN

[Diagram showing NASNet architecture]
DARTS: Continuous Search Space

A cell is a DAG consisting of sequence of N nodes:
- Each node $x^{(i)}$ is an intermediate result (tensor)
- Each directed edge $(i, j)$ is an operation (e.g. convolution)
- Two input nodes and a single output node

For each intermediate node $j$: $x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$

Output of the cell is obtained by reduction operation (addition / concatenation) to all intermediate nodes

**Continuous relaxation and optimization:**

$$\bar{o}^{(i,j)}(x) = \frac{\exp(\alpha^{(i,j)}_o)}{\sum_{o' \in \mathcal{O}} \sum_{o'' \in \mathcal{O}} \exp(\alpha^{(i,j)}_{o''})} o(x)$$

$\mathcal{O}$ is the set of operations and $\text{dim } \alpha^{(i,j)} = |\mathcal{O}|$

Architecture search:
Learn $\alpha = \{ \alpha^{(i,j)} \}$ (set of continuous mixing weights) and $o^{(i,j)} = \arg\max_{o \in \mathcal{O}} \alpha^{(i,j)}_o$
Bilevel Optimization

Joint learning of architecture $\alpha$ and network weights $w$.

Optimize for $\alpha^*$

\[
\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha)
\]

s.t. $w^*(\alpha) = \arg\min_w \mathcal{L}_{train}(w, \alpha)$

Approximation:

\[
\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \\
\approx \nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)
\]

Final architecture selection:

$\alpha^{(i,j)} = \arg\max_{\alpha \in \mathcal{O}} \alpha^{(i,j)}$

But, a node could be connected to too many predecessors!

DARTS employs pruning: retain only $k$ strongest connections
Experiment design

CIFAR-10 / PTB

DARTS

ImageNet / WikiText-2

Bilevel opt

Best cell architecture

Final cell architecture

Softmax

Input Image

Normal Cell

Reduction Cell

Small structure

Large structure

Small dataset

CIFAR-10 / PTB

Discard

Large dataset

Final evaluation

Large dataset (test set)

$M(\theta^*)$

$\theta^*$

$\alpha^*$

$w^*$

Final network training

Large dataset (training set)
Learned Cells

Normal cell (CIFAR-10)

Reduction cell (CIFAR-10)

Recurrent cell (PTB)

Note:
1. Exactly 2 incoming connections for CNN cells
2. Exactly 1 incoming node for RNN cells
3. The above are enforced through a pruning strategy
## Comparison

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Test Error (%)</th>
<th>Params (M)</th>
<th>+x (M)</th>
<th>Search Cost (GPU days)</th>
<th>Search Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception-v1 (Szegedy et al., 2015)</td>
<td>30.2</td>
<td>10.1</td>
<td>6.6</td>
<td>1448</td>
<td>manual</td>
</tr>
<tr>
<td>MobileNet (Howard et al., 2017)</td>
<td>29.4</td>
<td>10.5</td>
<td>4.2</td>
<td>569</td>
<td>manual</td>
</tr>
<tr>
<td>ShuffleNet 2× (g = 3) (Zhang et al., 2017)</td>
<td>26.3</td>
<td>–</td>
<td>~5</td>
<td>524</td>
<td>manual</td>
</tr>
<tr>
<td>NASNet-A (Zoph et al., 2018)</td>
<td>26.0</td>
<td>8.4</td>
<td>5.3</td>
<td>564</td>
<td>2000</td>
</tr>
<tr>
<td>NASNet-B (Zoph et al., 2018)</td>
<td>27.2</td>
<td>8.7</td>
<td>5.3</td>
<td>488</td>
<td>2000</td>
</tr>
<tr>
<td>NASNet-C (Zoph et al., 2018)</td>
<td>27.5</td>
<td>9.0</td>
<td>4.9</td>
<td>558</td>
<td>2000</td>
</tr>
<tr>
<td>AmoebaNet-A (Real et al., 2018)</td>
<td>25.5</td>
<td>8.0</td>
<td>5.1</td>
<td>555</td>
<td>3150</td>
</tr>
<tr>
<td>AmoebaNet-B (Real et al., 2018)</td>
<td>26.0</td>
<td>8.5</td>
<td>5.3</td>
<td>555</td>
<td>3150</td>
</tr>
<tr>
<td>AmoebaNet-C (Real et al., 2018)</td>
<td>24.3</td>
<td>7.6</td>
<td>6.4</td>
<td>570</td>
<td>3150</td>
</tr>
<tr>
<td>PNAS (Liu et al., 2018a)</td>
<td>25.8</td>
<td>8.1</td>
<td>5.1</td>
<td>588</td>
<td>~225</td>
</tr>
<tr>
<td>DARTS (searched on CIFAR-10)</td>
<td>26.7</td>
<td>8.7</td>
<td>4.7</td>
<td>574</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Perplexity</th>
<th>Params (M)</th>
<th>Search Cost (GPU days)</th>
<th>Search Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM + augmented loss (Inan et al., 2017)</td>
<td>91.5</td>
<td>87.0</td>
<td>28</td>
<td>–</td>
</tr>
<tr>
<td>LSTM + continuous cache pointer (Grave et al., 2016)</td>
<td>68.9</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>LSTM (Merity et al., 2018)</td>
<td>69.1</td>
<td>66.0</td>
<td>33</td>
<td>–</td>
</tr>
<tr>
<td>LSTM + skip connections (Melis et al., 2018)</td>
<td>69.1</td>
<td>65.9</td>
<td>24</td>
<td>–</td>
</tr>
<tr>
<td>LSTM + 15 softmax experts (Yang et al., 2018)</td>
<td>66.0</td>
<td>63.3</td>
<td>33</td>
<td>–</td>
</tr>
<tr>
<td>ENAS (Pham et al., 2018b)¹ (searched on PTB)</td>
<td>72.4</td>
<td>70.4</td>
<td>33</td>
<td>0.5</td>
</tr>
<tr>
<td>DARTS (searched on PTB)</td>
<td>71.2</td>
<td>69.6</td>
<td>33</td>
<td>1</td>
</tr>
</tbody>
</table>

¹ ENAS missing!
Summary

Main results
1. Proves gradient-based approach is possible and appropriate
2. Competitive results on accuracy
3. Outperforms all existing gradient-free methods in speed?
4. Demonstrated transferability from small to large datasets:
   a. CNN: CIFAR-10 to ImageNet
   b. RNN: PTB to WikiText-2

Advantages of DARTS
1. Fast and accurate
2. No controllers
3. General enough for CNN and RNN

Discussion points:
1. Why was ENAS not compared?
2. ENAS performance comes close despite being RL
3. Why Normal and Reduction cells?
ASAP: Architecture Search, Anneal and Prune

Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, Lihi Zelnik-Manor

AISTATS 2020
Problem Statement and Claims

- Adopts the same approach of DARTS (gradient-based)

- Argues that DARTS is not fast enough, the *post-training* pruning strategy is inefficient (*relaxation bias* [Xie et al., 2019])

- Gradual *during-training* pruning results in more efficient search

- In addition to continuity and differentiability of search space, ASAP advocates annealability for more efficient optimization.

- Claims to bring 1-4 GPU days of DART down to hours.
Method

A generalization of DARTS to annealable search space: DARTS + anneal and prune strategy

\[ o \in \mathcal{O} \quad x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}) \quad \tilde{o}^{(i,j)}(x; T) = \sum_{o \in \mathcal{O}} \Phi_o(\alpha^{(i,j)}; T) \cdot o(x) \]

\[ \Phi_o(\alpha^{(i,j)}; T) = \frac{\exp \left\{ \frac{\alpha^{(i,j)}}{T} \right\}}{\sum_{\alpha' \in \mathcal{O}} \exp \left\{ \frac{\alpha'_{i,j}}{T} \right\}} \quad \dim \alpha^{(i,j)} = |\mathcal{O}| \]

\( \Phi_o \) forms a uniform (distribution) for \( T \to \infty \) and sparse for \( T \to 0 \)

Annealing schedule: \( T(t) = T_0 \beta^t \)

Threshold policy: \( \Theta \equiv \theta_0 \)

Stopping condition: when only a single operation is left in \( \mathcal{O} \) for a \( (i, j) \)

Key to success is the balance between \( T(t) \) and \( \Theta \)

---

**Algorithm 1** ASAP for a single Mixed Operation

1: **Input**: Operations \( o_i \in \mathcal{O} \quad i \in \{1, \ldots, N\} \),
   - Annealing schedule \( T_i \),
   - Grace-temperature \( \tau \),
   - Threshold policy \( \theta_i \),
2: **Init**: \( \alpha_i \leftarrow 0 \quad i \in \{1, \ldots, N\} \),
3: while \( |\mathcal{O}| > 1 \) do
4:   Update \( \omega \) by descent step over \( \nabla_\omega \mathcal{L}_{\text{train}}(\omega, \alpha; T_i) \)
5:   if \( T_i < \tau \) then
6:     Update \( \alpha \) by descent step over \( \nabla_\alpha \mathcal{L}_{\text{val}}(\omega, \alpha; T_i) \)
7:     for each \( o_i \in \mathcal{O} \) such that \( \Phi_{o_i}(\alpha; T_i) < \theta_i \) do
8:       \( \mathcal{O} = \mathcal{O} \setminus \{o_i\} \)
9:     end for
10: end if
11: Update \( T_i \)
12: Update \( \theta_i \)
13: end while
14: return \( \mathcal{O} \)
Experiments

Follows the same experiment design of DARTS:

Fix outer structure $\Rightarrow$ search operations in a cell $\Rightarrow$ scale up outer structure $\Rightarrow$ train CNN

Architecture search in a small structure on CIFAR-10 took only 4.8 hours on single GPU.

Results on CIFAR-10 (no transfer)

Learned cells on CIFAR-10
# Experiments

## Transferability tests:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>CINIC-10 Error(%)</th>
<th>FREIBURG Error(%)</th>
<th>CIFAR-100 Error(%)</th>
<th>SVHN Error(%)</th>
<th>FMNIST Error(%)</th>
<th>ImageNet Error(%)</th>
<th>Search cost ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Known SotA</td>
<td>8.6</td>
<td>21.1</td>
<td>8.7</td>
<td>1.02</td>
<td>3.65</td>
<td>15.7</td>
<td>-</td>
</tr>
<tr>
<td>AmoebaNet-A</td>
<td>7.18</td>
<td>11.8</td>
<td>15.9</td>
<td>1.93</td>
<td>3.8</td>
<td>24.3</td>
<td>3150</td>
</tr>
<tr>
<td>NASNet</td>
<td>6.93</td>
<td>13.4</td>
<td>15.8</td>
<td>1.96</td>
<td>3.71</td>
<td>26.0</td>
<td>1800</td>
</tr>
<tr>
<td>PNAS</td>
<td>7.03</td>
<td>12.3</td>
<td>15.9</td>
<td>1.83</td>
<td>3.72</td>
<td>25.8</td>
<td>150</td>
</tr>
<tr>
<td>SNAS</td>
<td>7.13</td>
<td>14.7</td>
<td>16.5</td>
<td>1.98</td>
<td>3.73</td>
<td>27.3</td>
<td>1.5</td>
</tr>
<tr>
<td>DARTS-Rev1</td>
<td>7.05</td>
<td>11.4</td>
<td>15.8</td>
<td>1.94</td>
<td>3.74</td>
<td>26.9</td>
<td>1</td>
</tr>
<tr>
<td>DARTS-Rev2</td>
<td>6.88</td>
<td>10.8</td>
<td>15.7</td>
<td>1.85</td>
<td><strong>3.68</strong></td>
<td>26.7</td>
<td>1</td>
</tr>
<tr>
<td>ASAP</td>
<td><strong>6.83</strong></td>
<td><strong>10.7</strong></td>
<td><strong>15.6</strong></td>
<td><strong>1.81</strong></td>
<td>3.73</td>
<td>24.4</td>
<td><strong>0.2</strong></td>
</tr>
</tbody>
</table>
Experiments

Effect of *relaxation-bias*:
Summary

- A generalization of DARTS into annealable search space
- ASAP anneals and prunes the connection weights within the cell in a continuous manner
- Based on the insight that pruning during training reduces complexity and speeds up search.
- Theoretical results are available that enable good tradeoff between annealing schedule and threshold policy
- Achieves better training speed than DARTS while maintaining good accuracy

Discussion points:

1. The pruning strategy does not account for too many parents for a node in the cell. DARTS fixed this manually (k=2).
2. In spite of not fixing the above, all nodes in the learned cells have exactly two parents. This is a mystery!
References


