AutoML for Neural Machine Translation

Kevin Duh and Xuan Zhang
Johns Hopkins University
It's important to tune hyperparameters!

Histogram of BLEU scores for 700+ Swahili-English Neural Machine Translation (NMT) models

Note the large variance!
Hyperparameters

• Hyperparameters = Configurations of a model that are not updated in training

• Architectural hyperparameters:
  • # of layers
  • # of hidden units in feed-forward layer
  • # attention heads
  • Word embedding dimension

• Training pipeline hyperparameters:
  • # of subword units

• Optimizer hyperparameters:
  • Initial learning rate for ADAM, etc.
AutoML: Automated Machine Learning – what it might mean to different people

• For consumers: Democratization of ML
  • Upload own data, get ML model that can be plugged in application

• For developers: Reduce effort
  • Automate part of model building pipeline, more time for other priorities
  • Especially useful for optimizing models with speed-accuracy tradeoff

• For NMT researchers: Obtain state-of-the-art results
  • Fair comparison of methods

• For (some) ML researchers: Discover the next "Transformer"
AutoML: Automated Machine Learning – what it might mean to different people

• For **consumers**: Democratization of ML
  • Upload own data, get ML model that can be plugged in application

• For **developers**: Reduce effort
  • Automate part of model building pipeline, more time for other priorities
  • Especially useful for optimizing models with speed-accuracy tradeoff

• For **NMT researchers**: Obtain state-of-the-art results
  • Fair comparison of methods

• For **(some) ML researchers**: Discover the next "Transformer"
AutoML as an umbrella term

• Topics that might appear at an AutoML conference
  • Hyperparameter Optimization (HPO)
  • Neural Architecture Search (NAS)
  • Meta-Learning
  • Automated Reinforcement Learning (AutoRL)
  • Algorithm Selection
  • Systems for Machine Learning (SysML)
Goal of this tutorial

• **Motivate** the importance of proper hyperparameter tuning or architecture search

• **Explain a few popular methods in HPO and NAS** (focus in-depth on a few illustrative methods, then describe general categorizations)

• **Case study in NMT**: describe our experiences in applying AutoML, hope it serves as a reference for you

• We hope AutoML will someday be a useful part of your **toolbox**!
Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
6. Application to Neural Machine Translation (MT)
1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
   • Problem Formulation
   • Representative methods:
     • Bayesian Optimization
     • Grid/Random Search
     • Evolutionary strategies
     • Population-Based Training (PBT)
     • Hyperband
   • Generalizations
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
6. Application to Neural Machine Translation (MT)
Problem Definition: Hyperparameter Optimization (HPO)

Hyperparameter setting encoded as vector in $\mathbb{R}^d$

\[
\begin{align*}
3 &\rightarrow \text{# layers} \\
200 &\rightarrow \text{# units/layer} \\
1 &\rightarrow \text{optimizer type} \\
0.2 &\rightarrow \text{learning rate}
\end{align*}
\]

Train Model on dataset, then run diagnostics

$\mathbf{x} \rightarrow f(\mathbf{x})$

Accuracy (e.g. BLEU)

Find $\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x})$ with few function evaluations

*Clipart by Lea Duh*
Problem Definition: 
Hyperparameter Optimization (HPO)

Hyperparameter setting encoded as vector in $\mathbb{R}^d$

\[
\begin{align*}
3 & \rightarrow \text{# layers} \\
200 & \rightarrow \text{# units/layer} \\
1 & \rightarrow \text{optimizer type} \\
0.2 & \rightarrow \text{learning rate}
\end{align*}
\]

\[
\begin{align*}
\text{Train Model on dataset, then run diagnostics} \\
y = f(x)
\end{align*}
\]

Find $x^* = \text{argmax}_x f(x)$ with few function evaluations

Accuracy (e.g. BLEU)
Sequential Model-Based Optimization (SMBO)

Acquisition Function

Surrogate Model

Sample

Evaluate \( f(x) \)

Fit

\[
(x_1, \hat{f}(x_1)) \quad (x_2, \hat{f}(x_2)) \\
(x_3, \hat{f}(x_3)) \\
(x_i, \hat{f}(x_i))
\]
Bayesian Optimization

Expected Improvement

sample

evaluate $f(x)$

fit

$(x_1, \hat{f}(x_1))$
$(x_2, \hat{f}(x_2))$
$(x_3, \hat{f}(x_3))$
$(x_i, \hat{f}(x_i))$

$(x_p, f(x_p))$
Bayesian Optimization

**Gaussian Process:**
fit with uncertainty

**Expected Improvement:**
exploitation vs. exploration

Gaussian Process Regression

- Nonparametric / kernel methods

- $f_{GP}(x_{1:n})$ is jointly Gaussian; i.e. GP fits each $f_{GP}(x)$ w/ a Gaussian distribution.

- To predict $x_{new}$, GP compares how "similar" it is to $x_{1:n}$, which is measured by kernel.

- $\mu(x_{new})$ depends on the prior $\mu_0(x_{new})$ & $f(x_{1:n})$

Expected Improvement

Definition:

$$EI_n(x) := E_n \left[ (f(x) - f^*_n)^+ \right]$$

Expected Improvement

Definition:

\[ EI_n(x) := E_n \left[ \left( f(x) - f^*_n \right)^+ \right] \]

\[ EI(x) = (f^* - \mu) \Phi \left( \frac{f^* - \mu}{\sigma} \right) + \sigma \phi \left( \frac{f^* - \mu}{\sigma} \right) \]

where \( \phi, \Phi \) are the PDF, CDF of standard normal distribution.

Bayesian Optimization

Bayesian Optimization

Random / Grid Search

VS.

SMBO

Collect

Evaluate $f(x)$

Sample $x_p$

Evaluate $f(x)$

Fit $(x_p, f(x_p))$

Acquisition Function

Surrogate Model

Randomly sample $x_p$

$(x_1, f(x_1))$

$(x_4, f(x_4))$

$(x_i, f(x_i))$

$(x_3, f(x_3))$

$(x_2, f(x_2))$
Random Grid Search

Randomly sample $x_p$

Evaluate $f(x)$

Collect $(x_p, f(x_p))$

$(x_1, f(x_1))$
$(x_2, f(x_2))$
$(x_3, f(x_3))$
$(x_4, f(x_4))$

Easy to get parallelized

SMBO

Sample $x_p$

Evaluate $f(x)$

Acquisition Function

Surrogate Model

Fit $(x_p, f(x_p))$

$(x_1, f(x_1))$
$(x_2, f(x_2))$
$(x_3, f(x_3))$
$(x_4, f(x_4))$

Vs.
Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)

Evolutionary Strategy for HPO

1. Start with a population of “individuals”, each representing a hyperparameter setting.
2. The “fittest” ones (high $f(x)$) survive and produce offspring.
Evolutionary Strategy for HPO

1. Start with a population of “individuals”, each representing a hyperparameter setting
2. The “fittest” ones (high $f(x)$) survive and produce offspring

Generation 1
Evolutionary Strategy for HPO

1. Start with a population of “individuals”, each representing a hyperparameter setting
2. The “fittest” ones (high $f(x)$) survive and produce offspring

Generation 2
Estimating the search distribution

\[ \hat{\theta} = \arg \max_\theta \int f(x)N(x|\theta)dx \]
\[ \Delta \equiv \mathbb{E}[f(x)|\theta] \]

\[ \theta_0 = (\mu_0, \Sigma_0) \]

Mean update:
\[ \hat{\mu}_n = \hat{\mu}_{n-1} + \epsilon \mu \sum_{k=1}^{K} w(y_k)(x_k - \hat{\mu}_{n-1}) \]

Mean of previous generation

Vector from previous mean to \( x_k \)

Weight for individual \((x_k, y_k=f(x_k))\),

Better accuracy \( \rightarrow \) higher weight
Population Based Training (PBT)

From: Population Based Training of Neural Networks, Jaderberg et al. 2017
Population Based Training (PBT)

Figure. The objective function value of each worker over time.

From: Population Based Training of Neural Networks, Jaderberg et al. 2017
Go Beyond Blackbox HPO

• No need to train to completion every time.

• Performance early in training is highly correlated with performance late in training. (Dodge, et al. 2020.)

• Multi-fidelity Optimization:
  
  Use cheap approximations of the blackbox.

  e.g. fewer training steps.
Successive Halving (SHA)

-- multi-armed bandit algorithm to perform early stopping

From: automl.org
Successive Halving (SHA)

Two inputs:
Budget B, #configs N

B/n: resources allocated on average across the configurations

• Large N: small B/N, not enough training time
• Small N: large B/N, not enough configurations are evaluated
HyperBand

-- addresses the "n vs. B/n" problem by calling SHA multiple times with different n

<table>
<thead>
<tr>
<th>rung</th>
<th>N=81</th>
<th>N=27</th>
<th>N=9</th>
<th>N=6</th>
<th>N=5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>r</td>
<td>n</td>
<td>r</td>
<td>n</td>
</tr>
<tr>
<td>0</td>
<td>81</td>
<td>1</td>
<td>27</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>27</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>27</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>27</td>
<td>1</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HyperBand

![Graph showing regret versus wall clock time for Random Search and Hyperband, with a 20x speed up and 4x speed up indicated.](image-url)

From: automl.org
Generalizations

• There are many HPO methods, but they can be categorized along various aspects
  • Parallel vs Sequential
  • Search Algorithm vs Scheduler
  • Blackbox, Graybox, multi-fidelity
Generalization: Parallel vs Sequential

• Parallel vs Sequential:
  • Parallel: Evolutionary strategies, Population-based training
  • Sequential: Bayesian Optimization
  • What's best may depend on your compute setup & requirements

• All methods are iterative
  • All methods are about building on past experience in a HPO run
  • New research area: Meta-learning or transfer learning for HPO
    • Building on past experience from HPO runs on other problems
Generalization: Search Algorithm vs Scheduler

• Search algorithm: what to sample next (e.g. Bayes Opt vs CMA-ES)
• Scheduler: when to train a model, when to stop training (Hyperband)
• So these can be mixed and match!
  • HyperBand = Early stopping scheduler + Random Search
  • BOHB = Early stopping scheduler + Bayes Optimization
Generalization: Blackbox, Graybox, Multi-fidelity

• Blackbox methods don't look inside the model training process
• Graybox methods like Hyperband can improve HPO runtime
• Generally, multi-fidelity methods exploit approximations
  • Limit training time (analogous to Hyperband)
  • Training blackbox on smaller subset of data
  • Noisy measurements
  --> assume precise accuracy isn't needed
Section Summary

• Problem Formulation of HPO
• Representative methods:

  - Grid Search
  - Random Search
  - Bayesian Optimization
  - Population Based Training
  - CMA-ES
  - Successive Halving
  - HyperBand
  - Black-Box Search
  - Multi-Fidelity Search
  - Scheduler
Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
   - NAS vs HPO
   - Designing the NAS Search Space
   - NAS Search Strategy + Performance Estimation
     - Methods similar to HPO
     - One-shot NAS methods
4. Extension to Multiple Objectives
5. Evaluation
6. Application to Neural Machine Translation (MT)
## Hyperparameter Optimization (HPO) vs Neural Architecture Search (NAS)

<table>
<thead>
<tr>
<th></th>
<th>Hyperparameter Optimization (HPO)</th>
<th>Neural Architecture Search (NAS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine learning model</td>
<td>Neural Network, Random Forests, Support Vector Machines, etc.</td>
<td>Neural Network</td>
</tr>
<tr>
<td>Hyperparameters</td>
<td>Architectural:</td>
<td>Architectural</td>
</tr>
<tr>
<td></td>
<td>- #layer for neural net</td>
<td>- #layer, #dim</td>
</tr>
<tr>
<td></td>
<td>- tree depth for random forests</td>
<td>- &quot;Novel&quot; non-standard architectures</td>
</tr>
<tr>
<td></td>
<td>- kernel for support vector machine</td>
<td></td>
</tr>
<tr>
<td>Training Pipeline</td>
<td>Preprocessing, Data selection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Optimization:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- ADAM vs AdaGrad, Learning rate</td>
<td></td>
</tr>
<tr>
<td>Example of a discovered</td>
<td>4-layer encoder, 3-layer decoder, each with FFN of 512 dimensions</td>
<td>4-layer encoder: layer 1 has 512 dim, layer 2 has 1024 dim, layer 3 uses 12 heads rather than 8, etc.</td>
</tr>
<tr>
<td>Summary</td>
<td>General technique, course-grained but diverse hyperparameters</td>
<td>Focused technique on neural nets, fine-grained architectural space</td>
</tr>
</tbody>
</table>
Example of model discovered by NAS from: D. So, C. Liang, Q. Le. The Evolved Transformer (2019)
Three components to an NAS method

Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture $A$ from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of $A$ to the search strategy.
Three components to an NAS method

We'll discuss:

- Sequential vs. Cell-based
- Methods similar to HPO + Gradient-based
- Full train from scratch vs Weight share, One-shot, etc.

Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture $A$ from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of $A$ to the search strategy.

Search Space defined by sequential decisions

• Suppose we want feed-forward network with convolution layers
• Use a "controller" to predict hyperparameters in sequence

From: Zoph & Le. Neural Architecture Search with Reinforcement Learning, ICLR2017
Cell-based Search Space

• Focus search on smaller cells, which are stacked
• Example:
  • $V$ nodes per cell (e.g. Max $|V| = 7$)
  • Each node takes one of $L$ operations: 3x3 convolution, 1x1 convolution, 3x3 max-pool
  • Edges connect nodes, form Directed Acyclic Graph (DAG) starting from "in" to "out" node. (e.g. 21 edges max)
  • Encoding: 7x7 upper-triangular matrix + list of 5 operations. $2^{21} \times 3^5 = 510M$ unique cells

From: Ying et. Al. NAS-Bench-101: Toward Reproducible NAS
Cell-based Search Space (exercise)

# Adjacency matrix of the module
matrix=[[0, 1, 1, 1, 0, 1, 0],  # input layer
       [0, 0, 0, 0, 0, 0, 1],  # 1x1 conv
       [0, 0, 0, 0, 0, 0, 1],  # 3x3 conv
       [0, 0, 0, 0, 1, 0, 0],  # 5x5 conv (replaced by two 3x3's)
       [0, 0, 0, 0, 1, 0, 0],  # 5x5 conv (replaced by two 3x3's)
       [0, 0, 0, 0, 0, 0, 1],  # 3x3 max-pool
       [0, 0, 0, 0, 0, 0, 0]],  # output layer

# Operations at the vertices of the module, matches order of matrix
ops=[INPUT, CONV1X1, CONV3X3, CONV3X3, CONV3X3, MAXPOOL3X3, OUTPUT]]

From: Ying et. Al. NAS-Bench-101: Toward Reproducible NAS
Three components to an NAS method

We'll discuss:

Sequential vs. Cell-based

Methods similar to HPO + Gradient-based

Full train from scratch vs Weight share, One-shot, etc.

Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture $A$ from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of $A$ to the search strategy.

Search Strategy Options: HPO methods

• Sample x from NAS search space
• The rest we can use any HPO method:
  • Random search
  • Bayes Optimization
  • Evolutionary Strategy
  • Population-based Training
  • Hyperband
• Again we treat problem as a black box optimization
Search Strategy Options: Reinforcement Learning

• View exploration/exploitation in search space as a sequence of decisions
Search Strategy Options: Gradient-based

• DARTS: Differentiable Architecture Search (Liu, Simonyan, Yang; 2019)
  • addresses scalability issue in search + performance estimation by relaxing search space to be continuous
DARTS

• Let $O$ be set of candidate operations (e.g. convolution, max-pool, zero)
• For each edge $(i,j)$, we have a distribution

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

---

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge $(i, j)$

while not converged do

1. Update architecture $\alpha$ by descending $\nabla_{\alpha} \mathcal{L}_{val}(w - \xi \nabla_{w} \mathcal{L}_{train}(w, \alpha), \alpha)$
   $(\xi = 0$ if using first-order approximation$)$  # learn alpha on validation set
2. Update weights $w$ by descending $\nabla_{w} \mathcal{L}_{train}(w, \alpha)$  # fix alpha, standard training of parameters

Derive the final architecture based on the learned $\alpha$.  # pick argmax edges, retrain final model
Another one-shot NAS method: Once-for-All

• A single "supernet" is trained once
• Subnets x are sampled from supernet, and f(x) is measured without retraining x from scratch
• Progressive shrinking technique:
  • Potentially more representative subnets in supernet

From: Cai et. al. Once-for-all: Train one network and specialize it for efficient deployment. ICLR2020
Once-for-All applied to Transformers

1. Train a SuperTransformer by uniformly sampling SubTransformers with weight sharing
2. Collect Hardware Latency Datasets
3. Evolutionary Search
4. Train a Latency Predictor for each Hardware

From: Wang, et. al. HAT: Hardware-aware Transformers for Efficient NLP. ACL2020

Encoder Layer $m$

Elastic Layer
Num in Encoder:

Encoder Layer 2

Elastic Hidden Dim in FFN

Elastic Head Num
(Self Attention)

E.g. 4 heads

SuperTransformer

Decoder Layer $n$

Elastic Layer
Num in Decoder

Elastic Hidden Dim in FFN

Elastic Head Num
(En-Decoder Attention)

E.g. 2 heads

concat

Arbitrary Encoder-Decoder Attention

Elastic Embedding Dim

Elastic Embedding Dim

SubTransformer Architecture

Evolutionary Search Engine

Val Loss

Latency

Layer Num
Embed Dim
Heads Num

Latency Predictor

IoT
CPU
GPU
Pros & Cons of One-Shot NAS

• Pros:
  • Much faster than black-box search + performance estimation
  • Explore much larger architectural space

• Cons:
  • Difficult to know if the assumption of weight sharing is valid
  • Empirical results are mixed and unstable (some researchers may disagree)
  • Supernet needs to fit in memory

• NAS (one-shot & in general) is a very active research area – stay tuned!
Section Summary

We discussed:

- Sequential vs. Cell-based
- Methods similar to HPO + Gradient-based
- Full train from scratch vs Weight share, One-shot, etc.

Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture $A$ from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of $A$ to the search strategy.

Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
   • Why it's important
   • Pareto optimality
   • Example Multi-objective HPO/NAS methods
5. Evaluation
6. Application to Neural Machine Translation (MT)
When deploying models, we care about multiple objectives. But it's complex.
Quiz: How do these hyperparameters impact accuracy and speed?

• Architectural hyperparameters:
  • # of layers
  • # of hidden units in feed-forward layer
  • # attention heads
  • Word embedding dimension

• Training pipeline hyperparameters:
  • # of subword units

• Optimizer hyperparameters:
  • Initial learning rate for ADAM, etc.
Objectives one may care about

• Accuracy
  • BLEU, COMET, Human evaluation

• Inference speed
  • On GPU, on CPU, in batch or not
  • Throughput vs Latency

• Deployment resource consumption
  • Memory, disk, energy

• Training resource consumption
Motivation for Multiple Objectives

• IMHO, this is the strongest motivation for AutoML in deployment
  • While an engineer/researcher may develop good heuristics for tuning hyperparameters for accuracy alone, it is very difficult to reason through multiple interacting objectives

• Ideal future, where AutoML is part of everyone's toolkit
  • import AutoMLtool
  • A=search_space()
  • O=[accuracy(), speed(), memory()]
  • models = multi_objective_NAS(A, O)
How to define optimality for multi-objective?

![Graph showing multiple objectives with points A, B, C, D, E, F, and G.]
Definition: A point $p$ is weakly pareto-optimal iff there does not exist another point $q$ such that $F_k(q) > F_k(p)$ for all $k$.
Definition: A point $p$ is **weakly pareto-optimal** iff there does not exist another point $q$ such that $F_k(q) > F_k(p)$ for all $k$.
Definition: A point $p$ is **weakly pareto-optimal** iff there does not exist another point $q$ such that $F_k(q) > F_k(p)$ for all $k$. 

![Diagram showing points A, B, D, G, and C, with regions shaded in the first quadrant.](image)
Definition: A point $p$ is **pareto-optimal** iff there does not exist a $q$ such that $F_k(q) \geq F_k(p)$ for all $k$ and $F_k(q) > F_k(p)$ for at least one $k$.
Given a set of points, the subset of pareto-optimal points form the **Pareto Frontier**
Computing Pareto

• Pseudo-code:
  • Set N=[]
  • For p in ListOfSamples:
    • Set d = 0
    • For q in ListOfSamples:
      • For k in ListOfObjective, see if $F_k(q) > F_k(p)$. If yes, d+=1
      • If d=0, add p to N
  • Return N

• Basic implementation is $O(KN^2)$
  • $K = \#\text{objectives}, N= \#\text{samples}$
  • $O(K N\log N)$ is possible in two-objective case

• Generally, #pareto increases with K
Points can be ranked by successively peeling off the Pareto Frontier and recomputing.
Aside: Alternative to Pareto Optimality

• Combine multiple objectives into one

\[
\max_x \left[ f_1(x), f_2(x), \ldots, f_M(x) \right]
\]

Scalarization: \[
\max_x \left[ \sum_m \alpha_m f_m(x) \right] \quad \alpha_m \geq 0, \sum_{m=1}^{M} \alpha_m = 1
\]
Scalarization misses Pareto points that are not on Convex Hull

For more info, see tutorial: [https://www.cs.jhu.edu/~kevinduh/notes/duh11multiobj-handout.pdf](https://www.cs.jhu.edu/~kevinduh/notes/duh11multiobj-handout.pdf)
Incorporating Pareto into CMA-ES

\[ \hat{\theta} = \arg \max_{\theta} \int f(x) N(x|\theta) \, dx \]

\[ \triangleq \mathbb{E}[f(x)|\theta] \]

\[ \theta_0 = (\mu_0, \Sigma_0) \]

Mean update:

\[ \hat{\mu}_n = \hat{\mu}_{n-1} + \epsilon_\mu \sum_{k=1}^{K} \omega(y_k)(x_k - \hat{\mu}_{n-1}) \]

Mean of previous generation

Vector from previous mean to \( x_k \)

Weight for individual \( (x_k, y_k=f(x_k)) \),
Better Pareto rank \( \rightarrow \) higher weight
Example MT results from CMA-ES

From: Qin, Shinozaki, Duh. Evolutionary strategy based automatic tuning of NMT systems, IWSLT 2017
Multi-Objective Bayesian Optimization with Expected Hypervolume Improvement

Pareto Frontier
Multi-Objective Bayesian Optimization with Expected Hypervolume Improvement

Hypervolume Indicator
Multi-Objective Bayesian Optimization with Expected Hypervolume Improvement

Hypervolume Improvement (HVI)
Multi-Objective Bayesian Optimization with Expected Hypervolume Improvement

Objective function can be modeled as a multivariate Gaussian Process.

Expected Hypervolume Improvement:

\[ \alpha_{EHVI}(\mathbf{x}_{\text{cand}}) = \mathbb{E} \left[ HVI(f(\mathbf{x}_{\text{cand}})) \right] \]
Section Summary

• Pareto Optimality and multi-objective HPO/NAS
• Multi-objective is one of the strongest selling points of AutoML
  • Suppose Transformer-Big/Base doesn't fit your deployment scenario:

From: Wang, et. al. HAT: Hardware-aware Transformers for Efficient NLP. ACL2020
Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
   • Brief literature survey
   • Challenge of rigorous evaluation
   • Carbon footprint and broader issues
6. Application to Neural Machine Translation (MT)
Which HPO/NAS method is best?

• This question is difficult to answer, perhaps even ill-defined.
  • Depends on budget, evaluation metric, task
• We'll survey 4 papers that compare HPO & NAS on non-MT tasks, just to get a sense of the landscape
• We'll then describe competition result of the AutoML'22 MT benchmark.
• The message:
  • Evaluation of HPO/NAS methods is difficult due to computational constraints
  • The "best" solution for your problem will depend not just on the HPO/NAS method, but also on "best practices" for implementation (discussed later).
HPO comparison 1: Falkner, et. Al. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. ICML2018

• "Best" method depends on your budget
• Compare methods by fixing budget, or "anytime" performance

Kohavi96 Adult dataset: predict whether a person makes over 50k per year (features from Census)
HPO comparison 2: Zoller & Huber, Benchmark and Survey of Automated Machine Learning Frameworks, JAIR 2021

For datasets here, it seems:
- Some trends, e.g. Random Search is competitive, Grid search isn't
- But generally ranking is not consistent across datasets, variance is high
NAS Comparison 1: Yang et. al. NAS Evaluation is Frustratingly Hard, ICLR 2020

Object/scene classification data:

FLOWERS102
MIT67 (indoor scene)

CIFAR10 & CIFAR100, 60k 32x32 images

SPORT8
• Compare NAS with random sampling in same space (not random search)
• Improvements not large/consistent...
• Paper argues training protocol more important
### CIFAR-10: 32x32 pixel image, 10 classes, 60k samples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Published in</th>
<th>#Params (Millions)</th>
<th>Top-1 Acc(%)</th>
<th>GPU Days</th>
<th>#GPUs</th>
<th>AO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-110 [2]</td>
<td>ECVR16</td>
<td>1.7</td>
<td>93.57</td>
<td>-</td>
<td>-</td>
<td>Manually designed</td>
</tr>
<tr>
<td>PyramidNet [207]</td>
<td>CVPR17</td>
<td>26</td>
<td>96.69</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DenseNet [127]</td>
<td>CVPR17</td>
<td>25.6</td>
<td>96.54</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Go-Net (G=50) [30]</td>
<td>ICCV17</td>
<td>-</td>
<td>92.97</td>
<td>17</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Large-scale ensemble [25]</td>
<td>ICLR17</td>
<td>40.4</td>
<td>95.6</td>
<td>2,500</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>Hierarchical-EAS [19]</td>
<td>ICLR18</td>
<td>15.7</td>
<td>96.25</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-50 [189]</td>
<td>OIS2018</td>
<td>6.1</td>
<td>94.02</td>
<td>27</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>AmoebaNet-B (N=6, F=128)(+/−c=[26]</td>
<td>AAAI19</td>
<td>34.9</td>
<td>97.87</td>
<td>3,150</td>
<td>450 K40</td>
<td></td>
</tr>
<tr>
<td>AmoebaNet-B (N=6, F=32)(+/−c=[26]</td>
<td>AAAI19</td>
<td>2.8</td>
<td>97.45</td>
<td>3,150</td>
<td>450 K40</td>
<td></td>
</tr>
<tr>
<td>Lemonade [27]</td>
<td>ICLR18</td>
<td>3.5</td>
<td>97.6</td>
<td>8 Titan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENNA [149]</td>
<td>ICCV17</td>
<td>8.4</td>
<td>97.44</td>
<td>1 Titan Xp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENNA (more channels) [149]</td>
<td>ICCV17</td>
<td>54.14</td>
<td>97.79</td>
<td>0.65</td>
<td>1 Titan Xp</td>
<td></td>
</tr>
<tr>
<td>NASW [15]</td>
<td>ICLR17</td>
<td>7.1</td>
<td>95.53</td>
<td>22,400</td>
<td>800 K40</td>
<td></td>
</tr>
<tr>
<td>NASW + more filters [12]</td>
<td>CVPR17</td>
<td>37.4</td>
<td>96.35</td>
<td>230</td>
<td>800 K40</td>
<td></td>
</tr>
<tr>
<td>MetaQNN [23]</td>
<td>ICLR17</td>
<td>-</td>
<td>93.08</td>
<td>100</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>NASNet-A (7 Q 2048)(+/−c=[15]</td>
<td>CVPR18</td>
<td>87.6</td>
<td>97.6</td>
<td>2,000</td>
<td>500 P100</td>
<td></td>
</tr>
<tr>
<td>NASNet-A (6 700)(+/−c=[15]</td>
<td>CVPR18</td>
<td>3.3</td>
<td>97.95</td>
<td>2,000</td>
<td>500 P100</td>
<td></td>
</tr>
<tr>
<td>Block-QNN-Connection more filter [16]</td>
<td>CVPR18</td>
<td>33.3</td>
<td>97.92</td>
<td>696</td>
<td>32 1080Ti</td>
<td></td>
</tr>
<tr>
<td>Block-QNN-Depthwise, N=3 [16]</td>
<td>CVPR18</td>
<td>33.3</td>
<td>97.92</td>
<td>696</td>
<td>32 1080Ti</td>
<td></td>
</tr>
<tr>
<td>ENAS-macro [13]</td>
<td>ICLR18</td>
<td>38.0</td>
<td>96.12</td>
<td>0.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENAS-micro [13]</td>
<td>ICLR18</td>
<td>4.6</td>
<td>97.11</td>
<td>0.45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path-level EAS [139]</td>
<td>ICLR18</td>
<td>5.7</td>
<td>97.01</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path-level EAS(+/−c=[139]</td>
<td>ICLR18</td>
<td>5.7</td>
<td>97.01</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProxyNAS/SM-RL(+/−c=[132]</td>
<td>ICLR19</td>
<td>5.8</td>
<td>97.70</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPNAS [208]</td>
<td>ICCV19</td>
<td>5.76</td>
<td>96.99</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DARTS (fixed order)(+/−c=[17]</td>
<td>ICLR19</td>
<td>3.3</td>
<td>98.07</td>
<td>1.5</td>
<td>4 1080Ti</td>
<td></td>
</tr>
<tr>
<td>DARTS (almost order)(+/−c=[17]</td>
<td>ICLR19</td>
<td>3.3</td>
<td>97.99</td>
<td>4</td>
<td>4 1080Ti</td>
<td></td>
</tr>
<tr>
<td>sharpDARTS [178]</td>
<td>ArXiv19</td>
<td>3.6</td>
<td>97.00</td>
<td>0.8</td>
<td>2 1080Ti</td>
<td></td>
</tr>
<tr>
<td>P-DARTS(+/−c=[128]</td>
<td>ICLR19</td>
<td>3.4</td>
<td>97.50</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DARTS(large)(+/−c=[128]</td>
<td>ICLR19</td>
<td>10.5</td>
<td>97.75</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SETN [209]</td>
<td>ICLR19</td>
<td>4.6</td>
<td>97.31</td>
<td>1.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GDAS(+/−c=[154]</td>
<td>ICLR19</td>
<td>2.5</td>
<td>97.18</td>
<td>0.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNAS(moderate(+/−c=[155]</td>
<td>ICLR19</td>
<td>2.8</td>
<td>97.15</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baysas[210]</td>
<td>ICLR19</td>
<td>3.4</td>
<td>97.59</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProxyNAS-NSA(+/−c=[132]</td>
<td>ICLR19</td>
<td>5.7</td>
<td>97.92</td>
<td>0.1</td>
<td>1 1080Ti</td>
<td></td>
</tr>
<tr>
<td>PC-DARTS(+/−c=[211]</td>
<td>CVPR20</td>
<td>3.6</td>
<td>97.43</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIL-ENAS(+/−c=[132]</td>
<td>CVPR20</td>
<td>3.87</td>
<td>96.66</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMGG [212]</td>
<td>CVPR20</td>
<td>3.8</td>
<td>97.61</td>
<td>0.25</td>
<td>1 1080Ti</td>
<td></td>
</tr>
<tr>
<td>GDAS-NSAs[213]</td>
<td>CVPR20</td>
<td>3.54</td>
<td>97.27</td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ImageNet (subset): 224x224 pixel image, 1000 classes, 1 million samples

<table>
<thead>
<tr>
<th>Reference</th>
<th>Published in</th>
<th>#Params (Millions)</th>
<th>Top-1 Acc(%)</th>
<th>GPU Days</th>
<th>#GPUs</th>
<th>AO</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-152 [2]</td>
<td>CVPR16</td>
<td>230</td>
<td>70.92</td>
<td>50.51</td>
<td>-</td>
<td>Manually designed</td>
</tr>
<tr>
<td>PyramidNet [207]</td>
<td>CVPR17</td>
<td>116.4</td>
<td>70.85</td>
<td>5.3</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>SENet [126]</td>
<td>CVPR17</td>
<td>71.32</td>
<td>95.53</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DenseNet-201 [127]</td>
<td>CVPR17</td>
<td>76.35</td>
<td>78.54</td>
<td>94.46</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>MobileNetV2 [213]</td>
<td>CVPR18</td>
<td>6.9</td>
<td>74.7</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>GELF [17]</td>
<td>ICLR17</td>
<td>-</td>
<td>77.19</td>
<td>90.26</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>AmoebaNet-C((N=4,F=50)) [26]</td>
<td>AAAI19</td>
<td>6.4</td>
<td>75.71</td>
<td>92.4</td>
<td>3,150 K40</td>
<td></td>
</tr>
<tr>
<td>Hierarchical-EAS [19]</td>
<td>ICLR18</td>
<td>-</td>
<td>73.97</td>
<td>94.8</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>AmoebaNet-C((N=6,F=228)) [26]</td>
<td>AAAI19</td>
<td>155.3</td>
<td>83.16</td>
<td>96.3</td>
<td>3,150 K40</td>
<td></td>
</tr>
<tr>
<td>Gnas-Net [210]</td>
<td>CVPR20</td>
<td>6.5</td>
<td>77.18</td>
<td>93.3</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>NASNet-A(491056)</td>
<td>ICLR17</td>
<td>5.3</td>
<td>74.96</td>
<td>91.6</td>
<td>200 P100</td>
<td></td>
</tr>
<tr>
<td>NASNet-A(604022)</td>
<td>ICLR17</td>
<td>88.9</td>
<td>82.97</td>
<td>62.2</td>
<td>200 P100</td>
<td></td>
</tr>
<tr>
<td>Block-QNN [16]</td>
<td>CVPR18</td>
<td>91</td>
<td>80.15</td>
<td>45.2</td>
<td>96</td>
<td>32 1080Ti</td>
</tr>
<tr>
<td>Path-level EAS</td>
<td>ICLR18</td>
<td>74.6</td>
<td>91.9</td>
<td>8.3</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ProxysNAS(GPU) [132]</td>
<td>ICLR19</td>
<td>-</td>
<td>75.91</td>
<td>92.5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>ProxysNAS-RL(mobile) [132]</td>
<td>ICLR19</td>
<td>-</td>
<td>74.62</td>
<td>92.5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>MineA130</td>
<td>CVPR19</td>
<td>5.2</td>
<td>76.79</td>
<td>63.3</td>
<td>1,666</td>
<td></td>
</tr>
<tr>
<td>Efficient-B[104]</td>
<td>ICLR19</td>
<td>5.3</td>
<td>77.3</td>
<td>93.5</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Efficient-B(\text{T}=142)</td>
<td>ICLR19</td>
<td>66</td>
<td>84.97</td>
<td>1.1</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>FFNAS [208]</td>
<td>ICCV19</td>
<td>3.41</td>
<td>73.3</td>
<td>0.8</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

### NAS comparison 2: He, et. Al. AutoML: A Survey of the State-of-the-Art, 2021

- **Evolutionary**
- **Reinforcement Learning**
- **Gradient**
- **SMBO, e.g. Bayesian**
- **Random Search**

Many results! Different budgets...
Evaluation in HPO/NAS is extremely hard!

• Note previous papers focused on mostly on smaller datasets
• Evaluation is hard due to computational constraint:
  • Suppose it takes 1 week to train one model
  • Each HPO algorithm samples and trains 100 models at best
• Cannot do head-to-head comparison, repeated trials don’t know if an algorithm really works!
  • Li & Talwalkar (2019) Random search & Reproducibility for Neural Architecture Search: “Of the 12 papers published since 2018 at NeurIPS, ICML, and ICLR that introduce novel NAS methods, none are exactly reproducible.”
(Crazy) Solution: Tabular Benchmarks

- One-time fixed cost:
  - Run grid/random search, training MANY models on some dataset
  - Publish all \( \{x,f(x)\} \) pairs in a table

- HPO algorithm developers:
  - Experiment with HPO on finite universe
  - Can run repeated trials quickly
Tabular Benchmark for NMT (Zhang & Duh, TACL2020)

<table>
<thead>
<tr>
<th>Hyperparameter Type</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td># BPE Subword Units</td>
<td>1k, 2k, 4k, 8k, 16k, 32k, 50k</td>
</tr>
<tr>
<td># Transformer Layers</td>
<td>1, 2, 4, 6</td>
</tr>
<tr>
<td>Word embedding</td>
<td>256, 512, 1024</td>
</tr>
<tr>
<td># Hidden Units</td>
<td>1024, 2048</td>
</tr>
<tr>
<td># Attention Heads</td>
<td>8, 16</td>
</tr>
<tr>
<td>Initial Learning Rate for ADAM</td>
<td>3x10^{-4}, 6x10^{-4}, 10x10^{-4}</td>
</tr>
</tbody>
</table>

Total: 2245 Transformer models, trained on ~1550 GPU days; record BLEU, train/test time, etc. [https://github.com/Este1le/hpo_nmt](https://github.com/Este1le/hpo_nmt)
Diversity in dataset

For each dataset, we order hyperparameter configurations by BLEU, then compare these rankings across datasets.

Low Spearman’s correlation imply no single best set of Transformer model across datasets.
Diversity in dataset:
Hyperparameter importance by fANOVA, measuring BLEU variance when changing a specific hyperparameter value pairs

en-ja dataset

sw-en dataset
Evaluation philosophy: Find HPO methods that are robust over multiple datasets before applying to target real-world data
Interesting observations: (1) winner varies by budget. (2) high variance

RS: Random Search
BO: Bayesian Optimization
GB: Graph-based

Tabular Benchmark enables detailed run-time comparison
Multi-objective evaluation metrics

Example results on sw-en data, 700+ models in tabular benchmark, 14 pareto points

FTO: Fixed-Target, time to One pareto

FTA: Fixed-Target, time to All pareto

FBP: Fixed-Budget, # of Pareto achieved
(set budget = 200 models max)

For each method, plot mean and one standard deviation bounds over 100 random runs
AutoML 2022 Competition https://automl.cc

AutoML-Conference 2022
1st International Conference on Automated Machine Learning

AutoML-Conference: embracing open source, collaboration & reproducibility

AutoML Conference 2022

📍 Baltimore, US (co-located with ICML)
📅 July 25, 2022 July 27, 2022
Top performers in AutoML'22 Competition

• ESI Algiers and LAMIH/CNRS France – Evolutionary approach
  • Latin Hypercube Sampling for initial population
  • XGBRank for fitting $x \rightarrow f(x)$, then creating “surrogate function”
  • Find next generation by optimizing NSGA-II on surrogate function

• AutoML@Freiburg – Bayes Opt. approach, with transfer learning
  • Tree-structured Parzen Estimator (TPE) for Bayes Optimization
  • Transfer learning from multiple MT datasets
  • Define task similarity by how often similar hyperparameters perform well
Beyond tabular benchmarks?

• Surrogate benchmark:
  • Use external ML model to estimate f(x)
  • These can create infinitely many new "rows" in table

• Open questions:
  • How many \{x, f(x)\} pairs are needed to train an accurate surrogate?
  • Will the surrogate model introduce bias?
  • IMHO, I'm not convinced we can do this for complex and large tasks like Transformer hyperparameters for NMT.
Surrogate benchmark

- Zela, et. Al. Surrogate NAS Benchmarks, ICLR2022
- Argues that ranking of NAS methods are similar when comparing true benchmark to surrogate benchmarks (on different external models)

<table>
<thead>
<tr>
<th>Model</th>
<th>$R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LGBboost</td>
<td>0.892</td>
</tr>
<tr>
<td>XGBboost</td>
<td>0.832</td>
</tr>
<tr>
<td>GIN</td>
<td>0.832</td>
</tr>
<tr>
<td>NGBoost</td>
<td>0.810</td>
</tr>
<tr>
<td>$\mu$-SVR</td>
<td>0.709</td>
</tr>
<tr>
<td>MLP (Path enc.)</td>
<td>0.704</td>
</tr>
<tr>
<td>RF</td>
<td>0.679</td>
</tr>
<tr>
<td>$\epsilon$-SVR</td>
<td>0.675</td>
</tr>
</tbody>
</table>
Discussion: CO2e footprint and energy cost

• AutoML is basically trading human effort with computer time
• What is the cost of compute?
  • We may enjoy the convenience of AutoML, but we should be aware of the cost and potentially inefficiencies
  • To put things in perspective, let's discuss how different HPO/NAS compare in terms of CO2 footprint and energy cost
  • AutoML has the potential to have both positive and negative impact!
Estimating CO2e footprint

<table>
<thead>
<tr>
<th>Consumption</th>
<th>CO2e (lbs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Air travel, 1 person, NY↔SF</td>
<td>1984</td>
</tr>
<tr>
<td>Human life, avg, 1 year</td>
<td>11,023</td>
</tr>
<tr>
<td>American life, avg, 1 year</td>
<td>36,156</td>
</tr>
<tr>
<td>Car, avg incl. fuel, 1 lifetime</td>
<td>126,000</td>
</tr>
</tbody>
</table>

**Training one model (GPU)**

| NLP pipeline (parsing, SRL)                      | 39         |
| NLP pipeline (parsing, SRL) w/ tuning & experiments | 78,468     |
| Transformer (big)                                | 192        |
| Transformer (big) w/ neural arch. search         | 626,155    |

Table 1: Estimated CO2 emissions from training common NLP models, compared to familiar consumption.¹
Estimating CO2e footprint

Power consumption (kWh) from training a model

\[ p_t = \frac{1.58t \times (p_c + p_r + g p_g)}{1000} \]

Power Usage Effectiveness (PUE) - energy for infrastructure (cooling)

Avg power draw (watts) from CPU, RAM, #g GPUs

CO2e: CO2 equivalent emission (includes other greenhouse gases)

\[ \text{CO}_2\text{e} = 0.954p_t \]

EPA's estimate of avg CO2 (in lb per kWh) based on U.S. non-renewable vs renewable sources

Strubell et. al., Energy and Policy Considerations for Deep Learning in NLP, ACL2019
### Estimating CO2e footprint

<table>
<thead>
<tr>
<th>Model</th>
<th>Hardware</th>
<th>Power (W)</th>
<th>Hours</th>
<th>kWh·PUE</th>
<th>CO2e</th>
<th>Cloud compute cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2T\text{base}</td>
<td>P100x8</td>
<td>1415.78</td>
<td>12</td>
<td>27</td>
<td>26</td>
<td>$41–$140</td>
</tr>
<tr>
<td>T2T\text{big}</td>
<td>P100x8</td>
<td>1515.43</td>
<td>84</td>
<td>201</td>
<td>192</td>
<td>$289–$981</td>
</tr>
<tr>
<td>ELMo</td>
<td>P100x3</td>
<td>517.66</td>
<td>336</td>
<td>275</td>
<td>262</td>
<td>$433–$1472</td>
</tr>
<tr>
<td>BERT\text{base}</td>
<td>V100x64</td>
<td>12,041.51</td>
<td>79</td>
<td>1507</td>
<td>1438</td>
<td>$3751–$12,571</td>
</tr>
<tr>
<td>BERT\text{base}</td>
<td>TPUv2x16</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td>$2074–$6912</td>
</tr>
<tr>
<td>NAS</td>
<td>P100x8</td>
<td>1515.43</td>
<td>274,120</td>
<td>656,347</td>
<td>626,155</td>
<td>$942,973–$3,201,722</td>
</tr>
<tr>
<td>NAS</td>
<td>TPUv2x1</td>
<td>32,623</td>
<td></td>
<td></td>
<td></td>
<td>$44,055–$146,848</td>
</tr>
<tr>
<td>GPT-2</td>
<td>TPUv3x32</td>
<td>168</td>
<td></td>
<td></td>
<td></td>
<td>$12,902–$43,008</td>
</tr>
</tbody>
</table>

Table 3: Estimated cost of training a model in terms of CO2 emissions (lbs) and cloud compute cost (USD).\textsuperscript{7} Power and carbon footprint are omitted for TPUs due to lack of public information on power draw for this hardware.
AutoML can have both positive and negative impact on carbon footprint

To be fair, these NAS methods aren't optimizing for training cost, but the difference with those that do can be large. Also, see next slide for revised estimate.
Estimating carbon footprint, revisited

- Recommended reading if interested: Patterson, et. al. Carbon Emissions and Large Neural Network Training
  - It's challenging to estimate CO2e retrospectively; ideal for each paper author to measure it
  - Specific data center & time matters
  - Inference may take more energy in the aggregate than training/AutoML
- Note CO2e for Evolved Transformer is very different from previous papers!

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Parameters (B)</td>
<td>0.064 per model</td>
</tr>
<tr>
<td>Percent of model activated on every token</td>
<td>100%</td>
</tr>
<tr>
<td>Developer</td>
<td>Google Georgia</td>
</tr>
<tr>
<td>Datacenter of original experiment</td>
<td></td>
</tr>
<tr>
<td>When model ran</td>
<td>Dec 2018</td>
</tr>
<tr>
<td>Datacenter Gross CO2e/KWh (kg/KWh when it was run)</td>
<td>0.431</td>
</tr>
<tr>
<td>Datacenter Net CO2e/KWh (kg/KWh when it was run)</td>
<td>0.431</td>
</tr>
<tr>
<td>Datacenter PUE (when it was run)</td>
<td>1.10</td>
</tr>
<tr>
<td>Processor</td>
<td>TPU v2</td>
</tr>
<tr>
<td>Chip Thermal Design Power (TDP in Watts)</td>
<td>280</td>
</tr>
<tr>
<td>Measured System Average Power per Accelerator, including memory, network interface, fans, host CPU (W)</td>
<td>208</td>
</tr>
<tr>
<td>Measured Performance (TFLOPS/s)¹²</td>
<td>24.8</td>
</tr>
<tr>
<td>Number of Chips</td>
<td>200</td>
</tr>
<tr>
<td>Training time (days)</td>
<td>6.8</td>
</tr>
<tr>
<td>Total Computation (floating point operations)</td>
<td>2.91E+21</td>
</tr>
<tr>
<td>Energy Consumption (MWh)</td>
<td>7.5</td>
</tr>
<tr>
<td>% of Google 2019 total energy consumption (12.2 TWh = 12,200,000 MWh) [Goo20]</td>
<td>0.00006%</td>
</tr>
<tr>
<td>Gross tCO2e for Model Training</td>
<td>3.2</td>
</tr>
<tr>
<td>Net tCO2e for Model Training</td>
<td>3.2</td>
</tr>
<tr>
<td>Fraction of NAS Estimate in [Str19] (284 tCO2e)</td>
<td>0.011</td>
</tr>
<tr>
<td>Fraction of equivalent jet plane CO2e round trip San Francisco ↔ New York (~180 t; see Ap. A)</td>
<td>0.018</td>
</tr>
</tbody>
</table>

3.2x2200 = 7040 lbs
Section Summary

• Evaluation of HPO/NAS is non-trivial in two aspects
  • First, what do you want to look at?
    • Fixed budget, or anytime performance
    • What metric? What datasets?
  • Second, can you even run the evaluation in a rigorous fashion?
    • Tabular & Surrogate benchmark
    • NMT example
  • Awareness of CO2e footprint discussions, potential of AutoML for positive and negative impact
Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
6. Application to Neural Machine Translation (MT)
   - Hyperparameters that matter: Literature survey
   - Implementing AutoML in practice: case study
Hyperparameters

• Architectural hyperparameters:
  • # of layers
  • # of hidden units in feed-forward layer
  • # attention heads
  • Word embedding dimension

• Training pipeline hyperparameters:
  • # of subword units

• Optimizer hyperparameters:
  • Initial learning rate for ADAM, etc.
Optimizer and learning rate

Denkowski & Neubig. Stronger Baselines for Trustable Results in Neural Machine Translation, WNMT2017

Subword units

Ding et. Al., A Call for Prudent Choice of Subword Merge Operations in Neural Machine Translation, MT Summit 2019

---

### BLEU score of Transformer models with different BPE units, and delta between best and worst models (IWSLT2015) -- don’t use defaults!

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0.5k</th>
<th>1k</th>
<th>2k</th>
<th>4k</th>
<th>8k</th>
<th>16k</th>
<th>32k</th>
<th>δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar-en</td>
<td>30.3</td>
<td>30.8</td>
<td>30.6</td>
<td>30.5</td>
<td>30.4</td>
<td>29.8</td>
<td>28</td>
<td>27.5</td>
<td>3.3</td>
</tr>
<tr>
<td>cs-en</td>
<td>24.6</td>
<td>23.3</td>
<td>23.0</td>
<td>22.7</td>
<td>21.2</td>
<td>22.6</td>
<td>20.6</td>
<td>21.0</td>
<td>4.0</td>
</tr>
<tr>
<td>de-en</td>
<td>28.1</td>
<td>28.6</td>
<td>28.0</td>
<td>28.4</td>
<td>27.7</td>
<td>27.5</td>
<td>26.7</td>
<td>25.2</td>
<td>3.4</td>
</tr>
<tr>
<td>fr-en</td>
<td>28.8</td>
<td>29.8</td>
<td>29.6</td>
<td>29.3</td>
<td>28.7</td>
<td>28.5</td>
<td>27.5</td>
<td>26.6</td>
<td>3.2</td>
</tr>
<tr>
<td>en-ar</td>
<td>12.6</td>
<td>13.0</td>
<td>12.1</td>
<td>12.3</td>
<td>11.8</td>
<td>11.3</td>
<td>10.7</td>
<td>10.6</td>
<td>2.4</td>
</tr>
<tr>
<td>en-cs</td>
<td>17.3</td>
<td>17.1</td>
<td>16.7</td>
<td>16.4</td>
<td>16.1</td>
<td>15.6</td>
<td>14.7</td>
<td>13.8</td>
<td>3.5</td>
</tr>
<tr>
<td>en-de</td>
<td>26.1</td>
<td>27.4</td>
<td>27.4</td>
<td>26.1</td>
<td>26.3</td>
<td>26.1</td>
<td>25.8</td>
<td>23.9</td>
<td>3.5</td>
</tr>
<tr>
<td>en-fr</td>
<td>25.2</td>
<td>25.6</td>
<td>25.3</td>
<td>25.5</td>
<td>25.3</td>
<td>24.7</td>
<td>24.1</td>
<td>22.8</td>
<td>2.8</td>
</tr>
</tbody>
</table>

---

Variance from random restarts

![BLEU score vs BPE size](image)

(a) ar-en
Figure 1: Test BLEU for character and BPE translation as architectures scale from 1 BiLSTM encoder layer and 2 LSTM decoder layers (1×2+2) to our standard 6×2+8. The y-axis spans 6 BLEU points for each language pair.
Speed-accuracy tradeoff with Deep Encoder, Shallow Decoder

Pushing the limits with very deep layers

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>a:6L-6L</td>
<td>41.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>b:12L-12L</td>
<td>42.6</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>c:24L-12L</td>
<td>43.3</td>
<td>+</td>
<td>+</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>d:48L-12L</td>
<td>43.6</td>
<td>+</td>
<td>+</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>+</td>
</tr>
<tr>
<td>e:60L-12L</td>
<td>43.8</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>+</td>
</tr>
<tr>
<td>f:36L-36L</td>
<td>43.7</td>
<td>+</td>
<td>+</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>+</td>
</tr>
<tr>
<td>g:12L-60L</td>
<td>43.1</td>
<td>+</td>
<td>+</td>
<td>=</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3: BLEU comparison of different encoder and decoder layers (using ADMIN initialization, on WMT’14 EN-FR). In the matrix, each element (i,j) indicates if the model in row i significantly outperforms the model in column j (+), under-performs j (-), or has no statistically significant difference (=).

### Hyperparameter exploration, sequentially

Araabi & Monz, Optimizing Transformer for Low-Resource Neural Machine Translation

<table>
<thead>
<tr>
<th>ID</th>
<th>System</th>
<th>5k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Transformer-big</td>
<td>3.3</td>
</tr>
<tr>
<td>2</td>
<td>Transformer-base</td>
<td>8.3</td>
</tr>
<tr>
<td>3</td>
<td>2 + feed-forward dimension (2048 → 512)</td>
<td>8.8</td>
</tr>
<tr>
<td>4</td>
<td>3 + attention heads (8 → 2)</td>
<td>9.2</td>
</tr>
<tr>
<td>5</td>
<td>4 + dropout (0.1 → 0.3)</td>
<td>10.6</td>
</tr>
<tr>
<td>6</td>
<td>5 + layers (6 → 5)</td>
<td>10.9</td>
</tr>
<tr>
<td>7</td>
<td>6 + label smoothing (0.1 → 0.6)</td>
<td>11.3</td>
</tr>
<tr>
<td>8</td>
<td>7 + decoder layerDrop (0 → 0.3)</td>
<td>12.9</td>
</tr>
<tr>
<td>9</td>
<td>8 + target word dropout (0 → 0.1)</td>
<td>13.7</td>
</tr>
<tr>
<td>10</td>
<td>9 + activation dropout (0 → 0.3)</td>
<td><strong>14.3</strong></td>
</tr>
</tbody>
</table>
Architecture hyperparameter and model size

• Sockeye v2 Transformer implementation
• Model size count: https://github.com/Este1le/nparam
  - #BPE of source, target= sb, tb
  - #layers = sn, tn
  - #embed=e
  - FF #hidden=f
• Total
  = tn*(8e*e+7e+2ef+f)
  + sn*(4e*e+5e+2ef+f)
  + 4e+(sb*e+tb*(2e+1))
Additional references on NMT hyperparameters

• Britz et. Al. Massive Exploration of Neural Machine Translation Architectures, EMNLP 2017
• Dewangan, et. Al. Experience of neural machine translation between Indian languages, Machine Translation (2021)
• Lankford, Afli, Way. Transformers for Low-Resource Languages: Is Feidir Linn!. MT Summit 2021
Software Implementation of AutoML

• HPO/NAS algorithms are in general simple to implement.
• Challenge is the interface with the ML toolkit and the underlying computing infrastructure.

• Design considerations:
  • Automatically submit jobs
  • Automatically check job states
  • Automatically evaluate and collect results
  • Parallelization
  • Maximize the GPU utilization
  • Allow users to customize the AutoML runs by specifying arguments, e.g. #GPU, #configuration, #epochs
Existing AutoML Toolkits

Google Vizier

Ray Tune

Figure 1: Architecture of Vizier service: Main components are (1) Dangling work finder (restarts work lost to preemptions) (2) Persistent Database holding the current state of all Studies (3) Suggestion Service (creates new Trials), (4) Early Stopping Service (helps terminate a Trial early) (5) Vizier API (JSON, validation, multiplexing) (6) Evaluation workers (provided and owned by the user).

From: Google Vizier: A Service for Black-Box Optimization, Golovin et al. 2017
https://docs.ray.io/en/latest/
Use existing AutoML toolkits or Implement your own?

• Choice 1:
  Take an existing AutoML toolkit, and reimplement your training pipeline.

• Choice 2:
  Already have a training pipeline, e.g. Amazon Sockeye for MT, add an AutoML wrapper on top of it.

  It's worth implementing AutoML from scratch in this case.
Case Study: Amazon Sockeye with AutoML

• Amazon Sockeye:
  An open-source sequence-to-sequence framework for NMT built on PyTorch.
  https://github.com/awslabs/sockeye

• Sockeye-recipes (Duh et al.):
  Training scripts and recipes for the Sockeye toolkit.
  https://github.com/kevinduh/sockeye-recipes3

• Sockeye-recipes with AutoML:
  Automatic hyperparameter search with asynchronous successive halving on top of sockeye-recipes.
  https://github.com/kevinduh/sockeye-recipes3/tree/automl
Outline for Case Study

• Asynchronous Successive Halving (ASHA)
• Software design
• Use case
Recall: Successive Halving (SHA)

-- multi-armed bandit algorithm to perform early stopping
Asynchronous Successive Halving (ASHA)

• In the sequential SHA, the algorithm waits for all configurations in a rung to complete before promoting configurations to next rung.

• ASHA removes the bottleneck created by synchronous promotions.

• It would promote a configuration to next rung when
  • There's an idle worker.
  • There's a configuration that is secured a position in the top 1/p of this rung.

• Parallelization with maximal GPU utilization
Asynchronous Successive Halving (ASHA)

- ASHA promotes a configuration to next rung when there's a configuration that is secured a position in the top $1/p$ of this rung.

$p$: 2 (promote top $\frac{1}{2}$ to next rung)
Asynchronous Successive Halving (ASHA)

- ASHA promotes a configuration to next rung when there's a configuration that is secured a position in the top $1/p$ of this rung.

$p$: 2 (promote top $\frac{1}{2}$ to next rung)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>BLEU</th>
<th>Rung</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4.7</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2.3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4.6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8.2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3.4</td>
<td>6</td>
</tr>
</tbody>
</table>

- #configs: 10
  - rung 0: 5 configurations
  - rung 1: 1 configuration

- Not started, running, finished
Asynchronous Successive Halving (ASHA)

p: 2 (promote top \( \frac{1}{2} \) to next rung)

<table>
<thead>
<tr>
<th>Rung 0</th>
<th>Rung 1</th>
<th>Rung 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>#configs: 10</td>
<td>#configs: 5</td>
<td>#configs: 3</td>
</tr>
</tbody>
</table>

- **Rung 0**
  - BLEU: 5.0 7.5 4.7 6.2 2.3 4.6 3.2 8.2 3.4

- **Rung 1**
  - BLEU: 11.3 8.5 10.0
  - #configs: 5

- **Rung 2**
  - #configs: 3

Legend:
- Yellow: not started
- Green: running
- Red: finished
Pseudo-Code

**Input:** configurations `configs`, state checking time interval `t`,
minimum training checkpoints `r`, checkpoints within each rung `u`,
maximum training checkpoints `R`, reduction rate `p`, number of GPUs `G`

If `runtime % t == 0` do
  For each `config` do
    `state = check_state(config)`
    `react_to_state(config, state, r, R)`
  end
If ASHA is finished do
  Return
end
For each idle GPU do
  `candidate = get_candidate(configs, p)`
  `promote(candidate)`
  `submit_train(candidate, GPU, u)`
end
Pseudo-Code

Input: configurations $configs$, state checking time interval $t$, minimum training checkpoints $r$, checkpoints within each rung $u$, maximum training checkpoints $R$, reduction rate $p$, number of GPUs $G$

If runtime % $t == 0$ do
For each $config$ do
  $state = check\_state(config)$
  react\_to\_state($config$, $state$, $r$, $R$)
end
If ASHA is finished do
  Return
end
For each idle GPU do
  $candidate = get\_candidate(configs, p)$
  promote($candidate$)
  submit\_train($candidate$, $GPU$, $u$)
end
end

At each time step, we check the state of each config, and submit jobs to idle GPUs.
Pseudo-Code

**Input:** configurations $\textit{configs}$, state checking time interval $t$, minimum training checkpoints $r$, checkpoints within each rung $u$, maximum training checkpoints $R$, reduction rate $p$, number of GPUs $G$

If runtime $\% t == 0$ do

```
For each config do
  state = check_state(config)
  react_to_state(config, state, r, R)
end
```

If ASHA is finished do
  Return
end

For each idle GPU do
  candidate = get_candidate(configs, p)
  promote(candidate)
  submit_train(candidate, GPU, u)
end
Pseudo-Code

**Input:** configurations \textit{configs}, state checking time interval \textit{t}, minimum training checkpoints \textit{r}, checkpoints within each rung \textit{u}, maximum training checkpoints \textit{R}, reduction rate \textit{p}, number of GPUs \textit{G}

If runtime \% \textit{t} == 0 do

For each config do

\textit{state} = \textit{check\_state}(\textit{config})

\textit{react\_to\_state}(\textit{config, state, r, R})

end

If ASHA is finished do

Return

end

Find config candidates and submit training jobs.

For each idle GPU do

\textit{candidate} = \textit{get\_candidate}(\textit{configs, p})

\textit{promote}(\textit{candidate})

\textit{submit\_train}(\textit{candidate, GPU, u})

end
Pseudo-Code

**Input:** configurations \( \text{configs} \), state checking time interval \( t \),
minimum training checkpoints \( r \), checkpoints within each rung \( u \),
maximum training checkpoints \( R \), reduction rate \( p \), number of GPUs \( G \)

If runtime \( \% t == 0 \) do
  For each config do
    \( \text{state} = \text{check\_state}(\text{config}) \)
    \( \text{react\_to\_state}(\text{config}, \text{state}, r, R) \)
  end
If ASHA is finished do
  Return
end
For each idle GPU do
  candidate = get\_candidate(\text{configs}, p)
  promote(candidate)
  submit\_train(candidate, GPU, u)
end

It is done by reading the train log.

```python
for l in lines:
    if "Maximum number of not improved checkpoints" in l:
        return CONVERGED
    elif "CUDA error: all CUDA-capable devices are busy or unavailable" in l:
        return GPU_ERROR
    elif "CUDA out of memory" in l:
        return MEM_ERROR
    elif "OverflowError" in l:
        return MATH_ERROR
    elif "Best validation perplexity: inf" in l or "Train-ppl=nan" in l:
        return DIVERGED
    elif "Stale file handle" in l:
        return STORAGE_ERROR
    if "Training finished" in lines[0]:
        return SUCCESS
return RUNNING
```
Pseudo-Code

Input: configurations `configs`, state checking time interval `t`,
minimum training checkpoints `r`, checkpoints within each rung `u`,
maximum training checkpoints `R`, reduction rate `p`, number of GPUs `G`

If runtime % `t` == 0 do
  For each `config` do
    `state = check_state(config)`
    `react_to_state(config, state, r, R)`
  end
If ASHA is finished do
  Return
end
For each idle GPU do
  `candidate = get_candidate(configs, p)`
  `promote(candidate)`
  `submit_train(candidate, GPU, u)`
end
end

<table>
<thead>
<tr>
<th>State</th>
<th>Reaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUNNING</td>
<td>N/A</td>
</tr>
<tr>
<td>SUCCESS / CONVERGED</td>
<td>Submit valid job or Collect evaluation results</td>
</tr>
<tr>
<td>GPU ERROR</td>
<td>Submit again</td>
</tr>
<tr>
<td>MEM ERROR / DIVERGED</td>
<td>Delete job and add it to blacklist</td>
</tr>
</tbody>
</table>
Pseudo-Code

Input: configurations $\texttt{configs}$, state checking time interval $t$, minimum training checkpoints $r$, checkpoints within each rung $u$, maximum training checkpoints $R$, reduction rate $p$, number of GPUs $G$

If runtime $\% t == 0$ do
  For each config do
    $state = \text{check\_state}(\text{config})$
    $\text{react\_to\_state}(\text{config}, state, r, R)$
  end
  If ASHA is finished do
    Return
  end
  For each idle GPU do
    $\text{Get configs that are ready to move to next rung. (ASHA: no need to wait till all the configs in current run to finish.)}$
    $\text{candidate = get\_candidate(configs, p)}$
    $\text{promote(candidate)}$
    $\text{submit\_train(candidate, GPU, u)}$
  end
end
Pseudo-Code

Input: configurations $\texttt{configs}$, state checking time interval $t$, minimum training checkpoints $r$, checkpoints within each rung $u$, maximum training checkpoints $R$, reduction rate $p$, number of GPUs $G$

If runtime $\% t == 0$ do
  For each config do
    $state = \text{check\_state}(\text{config})$
    $\text{react\_to\_state}(\text{config}, state, r, R)$
  end
  If ASHA is finished do
    Return
  end
  For each idle GPU do
    $\textit{candidate} = \text{get\_candidate}(\texttt{configs}, p)$
    $\text{promote(}\textit{candidate})$
    $\text{submit\_train( candidate, GPU, r, u, R) }$
  end
end

Pick one from all the candidates. Random search or Bayesian Optimization.
Pseudo-Code

Input: configurations *configs*, state checking time interval *t*,
minimum training checkpoints *r*, checkpoints within each rung *u*,
maximum training checkpoints *R*, reduction rate *p*, number of GPUs *G*

If runtime % *t* == 0 do
  For each *config* do
    *state* = `check_state(config)`
    `react_to_state(config, state, r, R)`
  end
  If ASHA is finished do
    Return
  end
  For each idle GPU do
    *candidate* = `get_candidate(configs, p)`
    `promote(candidate)`
    `submit_train(candidate, GPU, r, u, R)`
  end
end

Submit a train job and let it run for
min(*r*, u*rung, *R*)-min(*r*, u*(rung-1), *R*) checkpoints
Implementation Challenges

• How to get the job state?
  We check the job log.

• How to automatically check the job state?
  We set up a timer running in a background thread.

• How to interact with the grid / GPU cluster?
  Besides job states, we also check GPU states.
  We debug carefully with possible errors.

• How to deal with failed jobs?
  We either resubmit it or delete it.
Example Run

```bash
(sockeye3) xzhang@test1:/exp/xzhang/sockeye-recipes3/autowl$ sh submit_run_autowl.sh
2022-09-07 19:41:02,733 Run ASHA with Arguments:
minimum number of checkpoints (r): 1
number of checkpoints per rung (u): 1
maximum checkpoints (R): 6
reduction rate (p): 2
number of GPUs (G): 4
2022-09-07 19:41:02,733 work directory: /exp/xzhang/sockeye-recipes3/egs/asha/spacel/run1
2022-09-07 19:41:02,733 Single-objective optimization: BLEU will be optimized.
config id to real id: {0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9}

num_avail_gpu 4
Obtaining the candidates .....      
Rung 0 candidates 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Rung 1 candidates set()
Rung 2 candidates set()
Rung 3 candidates set()
Picked candidate: 4
2022-09-07 19:41:02,825 qsub -l mem_free=200G,h_rt=10:00:00,num_proc=10,gpu=1 -q gpu.q -o /exp/xzhang/4_train.log.o -e /exp/xzhang/sockeye-recipes3/egs/asha/spacel/run1/job_logs/4_train_log.e -N ashat4
bes3/egs/asha/spacel/run1/hpms/4.hpm -e sockeye3
Your job 10022180 ("ashat4") has been submitted
```
Example Run

2022-09-07 19:52:10,840 Saved ASHA states to /exp/xzhang/sockeye-recipes3/egs/asha/space1/run1/ckpt.json

config id to real id: {0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9}

config 1 train_job_state: SUCCESS val_job_state: RUNNING train_gpu_state: NOTEXIST val_gpu_state: RUNNING
config 7 train_job_state: SUCCESS val_job_state: RUNNING train_gpu_state: NOTEXIST val_gpu_state: RUNNING
config 6 train_job_state: RUNNING val_job_state: NOTSTARTED train_gpu_state: RUNNING val_gpu_state: NOTEXIST
config 3 train_job_state: RUNNING val_job_state: NOTSTARTED train_gpu_state: RUNNING val_gpu_state: NOTEXIST

num_avail_gpu 0
2022-09-07 19:53:41,440 --------------------------
Rung 0:
Finished Jobs 2 4 8 9
Ids 2 4 8 9
BLEU 1.7 2.1 1.3 3.1

ASHA finished successfully. The best config is 6 with 8.3 BLEU score.
Review

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
6. Application to Neural Machine Translation (MT)
It's important to tune hyperparameters!

Histogram of BLEU scores for 700+ Swahili-English Neural Machine Translation (NMT) models

Note the large variance!
Hyperparameter Optimization (HPO)
Neural Architecture Search (NAS)

We discussed:

- Sequential vs. Cell-based
- Methods similar to HPO + Gradient-based
- Full train from scratch vs Weight share, One-shot, etc.

Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture $A$ from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of $A$ to the search strategy.

When deploying models, we care about multiple objectives. But it's complex.
Evaluation is hard, so Tabular Benchmark for NMT (Zhang & Duh, TACL2020)

<table>
<thead>
<tr>
<th>Hyperparameter Type</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td># BPE Subword Units</td>
<td>1k, 2k, 4k, 8k, 16k, 32k, 50k</td>
</tr>
<tr>
<td># Transformer Layers</td>
<td>1, 2, 4, 6</td>
</tr>
<tr>
<td>Word embedding</td>
<td>256, 512, 1024</td>
</tr>
<tr>
<td># Hidden Units</td>
<td>1024, 2048</td>
</tr>
<tr>
<td># Attention Heads</td>
<td>8, 16</td>
</tr>
<tr>
<td>Initial Learning Rate for ADAM</td>
<td>3x10^{-4}, 6x10^{-4}, 10x10^{-4}</td>
</tr>
</tbody>
</table>

Total: 2245 Transformer models, trained on ~1550 GPU days; record BLEU, train/test time, etc.  
https://github.com/Este1le/hpo_nmt

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Domain</th>
<th>#models</th>
</tr>
</thead>
<tbody>
<tr>
<td>zh-en</td>
<td>TED</td>
<td>118</td>
</tr>
<tr>
<td>ru-en</td>
<td>TED</td>
<td>176</td>
</tr>
<tr>
<td>ja-en</td>
<td>WMT</td>
<td>150</td>
</tr>
<tr>
<td>en-ja</td>
<td>WMT</td>
<td>168</td>
</tr>
<tr>
<td>sw-en</td>
<td>MATERIAL</td>
<td>767</td>
</tr>
<tr>
<td>so-en</td>
<td>MATERIAL</td>
<td>605</td>
</tr>
</tbody>
</table>
Use existing AutoML toolkits or Implement your own?

• Choice 1:
  Take an existing AutoML toolkit, and reimplement your training pipeline.

• Choice 2:
  Already have a training pipeline, e.g. Amazon Sockeye for MT, add an AutoML wrapper on top of it.

  It's worth implementing AutoML from scratch in this case.
Questions or Comments?