DATE 21
DESIGN, AUTOMATION & TEST IN EUROPE
01 – 05 February 2021 · virtual conference
The European Event for Electronic System Design & Test

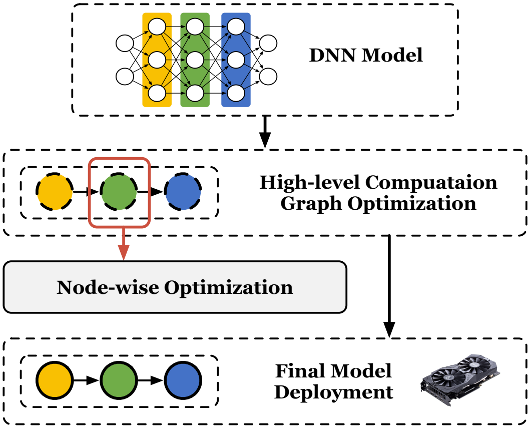
Deep Neural Network Hardware Deployment Optimization via Advanced Active Learning

Qi Sun, Chen Bai, Hao Geng, Bei Yu

香港中文大學
The Chinese University of Hong Kong

Background

Hardware deployment of deep neural networks



- General deployment framework.
- Support various hardware platforms and DNN models.
- A DNN model is represented as a graph.
- Layer-wise (node-wise) optimization, to determine the deployment configuration for each layer.
- The final model deployment configuration is the combination of the layers.

Background – Some Definitions

Deployment configuration

All of the deployment settings (*e.g.*, thread binding, tensor decomposition, *etc.*) to be determined are encoded as a feature vector, denoted as $\boldsymbol{x}$.

GFLOPS

Giga floating operations per second (GFLOPS) measures the number of floating-point operations conducted by the hardware per second.

Latency

Latency computes end-to-end model inference time and intuitively reflects the performance of model deployment.

Objective

Find the optimal $\boldsymbol{x}$, which has the best deployment performance, from the design space $\mathcal{D}$.

Background – Traditional active learning optimization

Initialization

- Randomly sample some configurations from the design space.
- Initialize an evaluation model (*e.g.*, XGBoost).

Iterative optimization

- Iteratively select a configruation according to the evaluation model and searching strategy
- Compile and deploy the configuration.
- Update the evaluation model.
- Stop until convergence.

Background

Current status

- Usually more than millions of candidate configurations in the design space.
- Slow compilation and depolyment processes.

Unsolved problems

- **Initialization** with underabundant information
- **Un-scalability** of the optimization process
- **Inaccuracy** of evaluation functions

Our Solution

Targets

- Improve data **diversity**.
- Improve model **scalability**.
- Improve model **accuracy**.

Proposed methods

- Batch Transductive Experimental Design
- Bootstrap-guided Adaptive Optimization

Initialization – Transductive Experimental Design (TED)

Maximize the intra-set diversity

$$x = \arg\max_{v \in \mathcal{V}} \frac{\|K_v\|^2}{k(v, v) + \mu}$$

- K: Euclidean distance matrix
- K_v is v's corresponding column in K
- $k(v, v)$ is v's diagonal entry in K, μ is a coefficient
- Iteratively select the configuration point which has the largest distance to other configurations
- The selected points are the **initialization set** for the evaluation model

Initialization – Batch TED

Batch method

- Computing distance matrix is very slow, even impossible.
- Sample a batch of sets from the design space and compute the distance matrix for each set.

Initialization – Batch TED

Batch method

- Computing distance matrix is very slow, even impossible.
- Sample a batch of sets from the design space and compute the distance matrix for each set.

Algorithm 2 Batch Transductive Experimental Design – **BTED**$(\mathcal{V}, \mu, M, m, B)$

Require: $(\mathcal{V}, \mu, M, m, B)$, where $\mathcal{V}$ is the un-sampled configuration set, μ is the normalization coefficient, B is the batch size, M is the number of randomly sampled points and m is the number of points to be sampled as the initial set.

Ensure: Newly sampled configuration set $\mathcal{X}$.

1: **for** $b = 1 \rightarrow B$ **do**
2:     Randomly sample a set $\mathcal{V}_b$ from $\mathcal{V}$, with $|\mathcal{V}_b| = M$;
3:     $\tilde{\mathcal{X}}_b \leftarrow$ **TED**$(\mathcal{V}_b, \mu, m)$;
4: **end for**
5: Temporal union set $\tilde{\mathcal{X}}_U = \tilde{\mathcal{X}}_1 \cup \tilde{\mathcal{X}}_2 \cup \cdots \cup \tilde{\mathcal{X}}_B$;
6: $\mathcal{X} \leftarrow$ **TED**$(\tilde{\mathcal{X}}_U, \mu, m)$; **return** Newly sampled configuration set $\mathcal{X}$;

Initialization – Batch TED

Batch method

- Computing distance matrix is very slow, even impossible.
- Sample a batch of sets from the design space and compute the distance matrix for each set.

Algorithm 2 Batch Transductive Experimental Design – **BTED**($\mathcal{V}, \mu, M, m, B$)

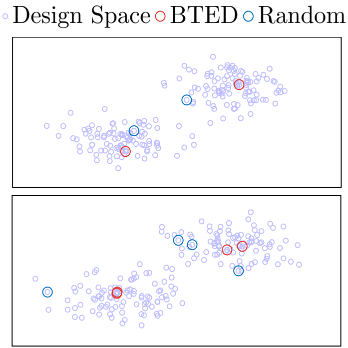
Require: ($\mathcal{V}$, μ, M, m, B), where $\mathcal{V}$ is the un-sampled configuration set, μ is the normalization coefficient, B is the batch size, M is the number of randomly sampled points and m is the number of points to be sampled as the initial set.

Ensure: Newly sampled configuration set $\mathcal{X}$.

```
for b = 1 → B do
    Randomly sample a set V_b from V, with |V_b| = M;
    X̃_b ← TED(V_b, μ, m);
end for
Temporal union set X̃_U = X̃_1 ∪ X̃_2 ∪ ··· ∪ X̃_B;
X ← TED(X̃_U, μ, m); return Newly sampled configuration set X;
```

- **for** $b = 1 \rightarrow B$ **do**
  - Randomly sample a set $\mathcal{V}_b$ from $\mathcal{V}$, with $|\mathcal{V}_b| = M$;
  - $\tilde{\mathcal{X}}_b \leftarrow$ **TED**$(\mathcal{V}_b, \mu, m)$;
- **end for**
- Temporal union set $\tilde{\mathcal{X}}_U = \tilde{\mathcal{X}}_1 \cup \tilde{\mathcal{X}}_2 \cup \cdots \cup \tilde{\mathcal{X}}_B$;
- $\mathcal{X} \leftarrow$ **TED**$(\tilde{\mathcal{X}}_U, \mu, m)$; **return** Newly sampled configuration set $\mathcal{X}$;

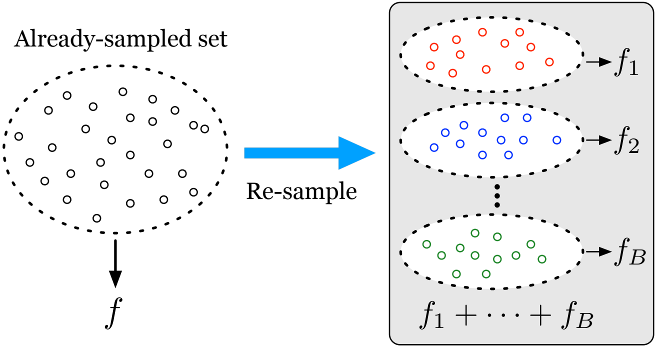
- Example



Iterative Opt. – Bootstrap-guided Adaptive Opt. (BAO)

Bootstrap re-sampling

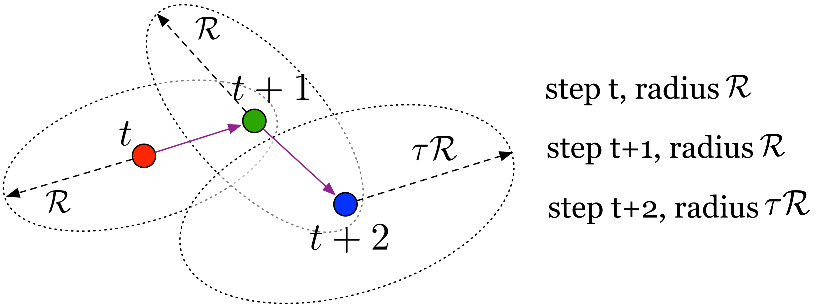
- Randomly re-sample a batch of sub-sets from the already-sampled configuration set.
- Build new evaluation functions for each of these re-sampled sub-sets.
- The final evaluation function is built as the summation of the evaluation functions of these re-sampled sub-sets.



Iterative Opt. – Bootstrap-guided Adaptive Opt. (BAO)

Adaptive Sampling

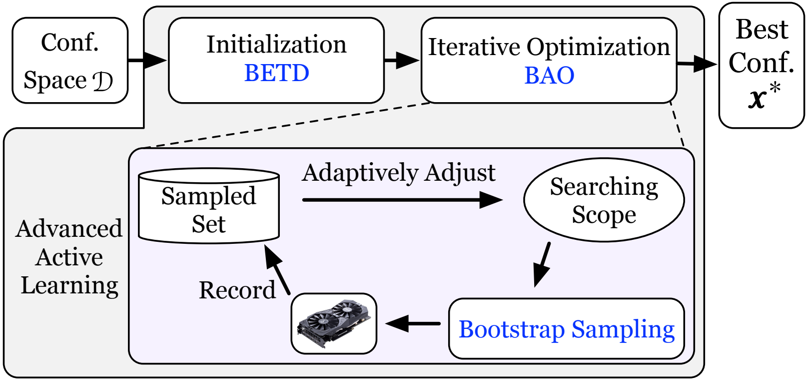
- Adjust the searching space (neighborhood of the previously-sampled point) adaptively in each optimization step.
- If the relative performance improvement is satisfying, we will keep the size of the searching space.
- Otherwise, we will enlarge the searching space.



Our Framework

Advanced active learning

- Batch transductive experimental design
- Bootstrap-guided adaptive optimization



Experimental Settings

Platform

- Intel(R) Xeon(R) E5-2680 v4 CPU@ 2.40GHz
- NVIDIA GeForce GTX 1080Ti GPU, CUDA 9.0.176

Benchmark

AlexNet, ResNet-18, VGG-16, MobileNet-v1, and SqueezeNet-v1.1

Criterion

- Inference latency
- Giga floating operations per second (GFLOPS)

Baseline

- AutoTVM

Convergence

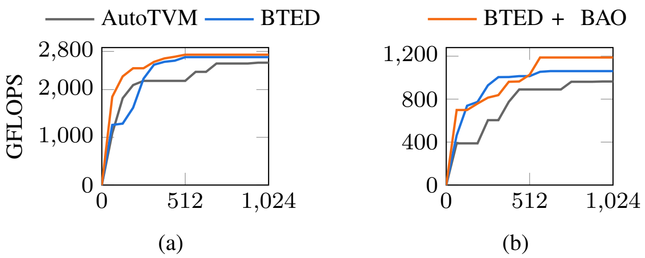


Fig. 4 Convergence trends of GFLOPs for the first 2 layers of MobileNet-v1, (a) the first layer, (b) the second layer.

Sampled Configurations and GFLOPS

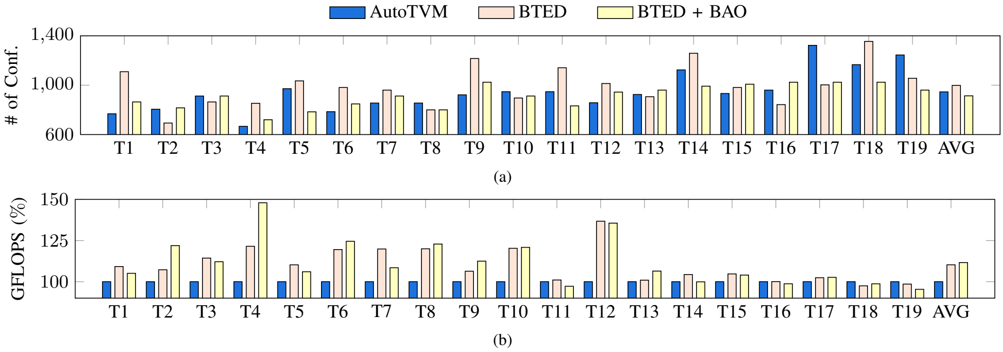


Fig. 5 The number of sampled configurations and GFLOPS values of MobileNet-v1. AVG represents the average results of the 19 tasks.

Results

Table: Comparisons of End-to-end Model Inference Latency and Variance

Model	AutoTVM		BTED				BTED + BAO			
	Latency (ms)	Variance	Latency (ms)	Δ (%)	Variance	Δ (%)	Latency (ms)	Δ (%)	Variance	Δ (%)
`AlexNet`	1.3639	0.1738	1.3373	- 1.95	0.2246	+29.23	1.3304	**- 2.46**	0.0711	**-59.09**
`ResNet-18`	1.8323	0.4651	1.7935	- 2.12	0.4487	- 3.53	1.7519	**- 4.39**	0.3848	**-17.27**
`VGG-16`	6.5176	2.3834	5.6808	-12.84	0.6574	-72.42	5.6183	**-13.80**	0.3617	**-84.82**
`MobileNet-v1`	1.0597	0.9290	0.8738	-17.54	0.5398	-41.89	0.7621	**-28.08**	0.0674	**-92.74**
`SqueezeNet-v1.1`	0.8697	1.1208	0.7436	-14.50	0.5533	-50.63	0.6920	**-20.43**	0.1709	**-84.75**
Average	2.3286	1.0144	2.0858	- 9.79	0.4848	-27.85	**2.0309**	**-13.83**	**0.2112**	**-67.74**

Thank You