Machine Learning and Computer Systems Notes

Zhongzhu Zhou Ph.D. reading

Jun 2021

Contents

1 Introduction

This is the note for my Ph.D. reading materials. After *two* years' trials in the notes in a 'word' document. I find it inefficient in summarizing and citing them in the further latex paper works. Thus, I begin using latex to record my paper reading notes.

Because it's not a Note for learning the full knowledge, I only store some key points or some points I may forget easily.

2 Machine Learning

2.1 Models

2.1.1 RNN

2.2 Meta-Learning

2.3 NAS

2.3.1 one shot learning

2.3.2 few-shot Neural Architecture Search

Recently, one-shot NAS substantially reduces the computation cost by training only one supernet- work, a.k.a. supernet, to approximate the perfor- mance of every architecture in the search space via weight-sharing.

Therefore, we can randomly choose which compound edge(s) to split and focus on how many com- pound edge(s) to split. In this work, we pre-define a training time budget T . If the total training time of supernet and all currently trained sub-supernets exceeds T , we will stop the split to avoid training more sub-supernets. Generally, T is set to be twice of one-shot supernet training time.

Insight 1: When shall we stop? Use a GAP. Insight 2: Why we always split the NN in the begining instead of during the experiments (DFS or BFS?) How to search efficently.

What can we split first? Choose the best then the edge.

2.4 Transformer

The background of transformer is seq2seq and attention.

2.4.1 seq 2 seq

[**?**] introduce

2.4.2 attention

2.4.3 AutoFormer: Searching Transformers for Visual Recognition

Previous works on designing vision transformers are based upon manual crafting, which heavily relies on human expertise and typically requires a deal of trial-and- error [1, 2, 3],.

we propose a supernet training strategy called weight entanglement dedicated to transformer architecture.

we propose a supernet training strategy called weight entanglement dedicated to transformer architecture. The central idea is to enable different transformer blocks to share weights for their common parts in each layer. An update of weights in one block will affect all other ones as a whole,

This strategy is different from most one-shot NAS methods [4, 5, 6]

it allows a large number of subnets in the super-net to be very well-trained, such that the performance of these subnets with weights inherited from the supernet are comparable to those retrained from scratch.

To tackle the challenges, we construct a large search space covering the main changeable dimensions of trans- former, including embedding dimension, number of heads, query/key/value dimension, MLP ratio, and network depth.

To address the efficiency issue, inspired by BigNAS [7] and slimmable networks [7, 8]

It allows a large number of subnets in the super- net to be very well-trained, such that the performance of these subnets with weights inherited from the supernet are comparable to those retrained from scratch.

We perform a evolutionary search with a model size constraint over the well-trained supernets to find promising transformers.

Given a 2D image, we first uniformly split it into a se- quence of 2D patches. A transformer encoder consists of alternating blocks of multihead self-attention (MSA) and multi-layer perceptron (MLP) blocks. LayerNorm (LN) [2] is applied before ev- ery block, and residual connections after every block.

where W is the weight of the supernet, W is shared across all the architecture candidates Since it is impossible to enumerate all the architectures $\alpha in A$ for evaluation, prior works resort to random search [34, 4], evolution algorithms [43, 16] or reinforcement learning [40, 48] to find the most promising one.

directly apply one-shot NAS for transformer search fol- lowing classical weight sharing strategy [16], using differ- ent weights for different blocks in each layer, because of the slow convergence and unsatisfactory performance. The reasons: 1. The reason might be that the independent training of transformer blocks results in the weights being updated by limited times. 2. The perfor- mances of subnets inheriting weights from the one-shot supernet, trained under classical weight sharing strategy, are far below their true performances of training from scratch.

The underlying reason is that homogeneous blocks are structurally compatible, such that the weights can share with each other. During implementation, for each layer, we need to store only the weights of the largest block among the n homogeneous candidates. The remaining smaller build- ing blocks can directly extract weights from the largest one.

During training, all possible subnets are uniformly sampled, and the corresponding weights are updated.

Such partition allows the search algorithm to concentrate on finding models within a specific parameter range, which can be specialized by users according to their available resources and application requirements.

For mutation, a candidate mutates its depth with probability Pd first. Then it mutates each block with a probability of Pm to produce a new architecture.

my comprehension: 1. For the weight entanglement, can we select several blocks to cobine them as a more effective subsupernets?

3 Virtualization

4 Computer System

4.1 MAEE - Multiple Applications Execution time Estimation

Insigts:

- make a open-source dataset for the applications
- Design an accurate model for co-running them with the fact of time shifting. Esitimate the future running time.
-

4.1.1 perf-tools

- perf
- iostat
- vmstat
- ifstat

perf stat Run a command and gather performance counter statistics
-I msecs

-a, –all-cpus system-wide collection from all CPUs (default if no target is specified)

-x SEP, –field-separator SEP print counts using a CSV-style output to make it easy to import directly into spreadsheets. Columns are separated by the string specified in SEP.

-e a symbolic event name (use perf list to list all events) a raw PMU event (eventsel+umask) in the form of rNNN where NNN is a hexadecimal event descriptor.

-p, –pid stat events on existing process id (comma separated list)
perf list

cpu/t1=v1[,t2=v2,t3 ...]/modifier [Raw hardware event descrip (see 'man perf-list' on how to encode it)

4.1.2 Job placement using reinforcement learning in GPU virtualization environment

This study defines the resource utilization history of applications and proposes a reinforcement learning-based job placement technique, which uses it as an input. For resource utilization history learning, a deep reinforcement learning model (DQN) is used.

4.1.3 Sequence-to-sequence models for workload interference prediction on batch processing datacenters

Since the average utilization is estimated to be below 50% [9, 10] The principal problem when co-locating resource-sharing ap- plications is to ensure that competition will not ruin their QoS,

Since interference creates a new characteristic footprint for each set of concurrent applications. Applications can be profiled in isolation in order to characterize the requirements claimed during their execution. Thus, there is the need to predict application resource demands and interference without the burden of running all possible combinations of applications.

Therefore, they produce a single value global prediction estimate, instead of a sequence of predictions over time. Most classic machine learning approaches for this problem, such as [11, 12], do not consider the temporal dimension of executions.

Our model employs two Gated Recurrent Units (GRU) [9] as building blocks; one GRU processes the trace signal of the incoming applications and passes the processed information to the other GRU, which outputs the expected resources of the co- located applications over time.

Intel HiBench [10], the IBM SparkBench [11] and the Databricks Spark-Perf benchmarks [12]

- A novel use of Recurrent Neural Networks that estimates the monitored metrics of two co-scheduled applications $a \wedge b$ from the information of a and b gathered running the applications in isolation.

- A novel feature, percentage completion time, for estimat- ing the completion time of co-scheduled applications. This feature improves predictions made by using the standard stopping criteria based on the end of sequence feature.
- A comprehensive evaluation of the method against other relevant machine learning approaches. We show the advan- tages of our method, which are especially noticeable in two cases: when co-located executions have different lengths, and when co-scheduling heavily impacts the execution time of the applications due to high interference.

The way in which st is computed is what mostly differentiates RNN models, such as Gated Recurrent Units [9] (GRU) and Long Short- Term Memory [13] (LSTM).

The decoder reads the encoded vector and produces a new sequence, but instead of initializing the hidden state of the decoder with zeros, it is initialized to the last hidden state of the encoder RNN

the context vector is computed as a matrix vector product, which can be interpreted as a weighted sum of the columns of E. The encoder generate this matrix E directly and multiply matrix E with the input, last state and finally use E and vt to generate a fixed size sequences. The input of decoder are context vector generated by the input vector, last hiden state input,

EOS. This feature vector takes value 0 at every time step except the last one, where it takes value 1. This vector simply tells the decoder when both applications finish.

In that scenario, EOS is just a special word that stops the decoding process, which means that if the decoder emits the ‘ ‘EOS’ ’ symbol the decoding process is stopped. In our setting the EOS is a new feature that takes a real value at every time step.

We denote by PCFa and PCFb the percentage completion features for input sequences a and b, respectively. Both features contain at time t how much of the workload has been completed until t, expressed as a percentage. For a given sequence s of length N, we define PCFs = (1 $\cdot$ 100, 2 $\cdot$ 100,..., N $\cdot$ 100). Notice that, by construction, PCF features contain monotonically increasing values that must finish with value 100. Nevertheless the rate of

increment at every time step will depend on the overall number of time steps of the sequence.

vmstat, iostat, ifstat and perf. The dataset used in the experiments contains traces generated by a variety of micro-benchmarks (workloads). The workloads used have been extracted from different suites: HiBench [10], Spark-perf [12] and SparkBench [11].

The author have three other model - basline model - ioslated prediction. The degradation of quality for long sequence prediction is a known issue found

Since we do not know in advance how long co-scheduled applications can take to finish, we have decided to treat the length of the generated trace as a hyperparameter to be adjusted. The different criteria tested are as follows:

applications are predicted to finish the larger the expected error. The box plot shows results for the criterion argmax PC sum

then errors are large because the stopping criteria is fired before the co-scheduled applications might finish. Errors decrease as k increases up to a point where the errors start to increase.

dynamic provisioning and job placement. predict whether there will be a change in the workload trace that needs some hardware decision.

Works like [13, 14] use neural networks that take as input resource usage in a given time window with the goal of predicting the future resource requirements for the workloads. In authors use differential evolution as a means to train the models, whereas in they use standard gradient based algorithms.

In [15], RNNs are used to predict cloud resource requests of Google cloud CPU and RAM requests, results are compared against ARIMA forecasting, achieving lower error with the RNN. they do not take into account the degradation of the applications under co-scheduled scenarios or the challenges that appear when a full time series is meant to be predicted from more than a input trace signal.

[16] it then passes through a k-Nearest Neighbor in order to choose the most representative expert. Finally, the application is run with different

data sizes in order to tune the function parameters to determine the best fit for this application.

SVD and PQ-reconstruction [29]. In Paragon, the profiling across pairs is limited to the first minute due to time constraints, without covering applications with different execution phases and behaviors over time

properly ensure resource availability for specific jobs. Such a solution tends to overresource applications for solving the interference problem, thus becoming subject to machine under-utilization. Another interesting work dealing with protection against in- terference is Stay-Away by [17]

In [18] the authors present a convolutional neural network named PRIONN. The model predicts run-time and IO (bytes read and total bytes written) of applications based on the source code in the input script.

In [19] Aloja-ML is presented as a framework for character- ization and knowledge discovery in Hadoop deployments. The authors present different methods for execution time prediction based on hardware resources, workload type and input data size.

Insights:

1. not consider some applications are running in the nodes. a^b represents an execution of the co-located pair. $a^{\rightarrow} \wedge b$ For the percentage completion time, it's hard to know whether the

2. Can we use a unsupervised learning algorithm for this problem? to do a classification for the program? Like the roofline model? I can try this.

3. No one use a SPEC HPC benchmark.

4. The author do not consider other time-series model as comparison

5. Can we abandon the EOS or PC and focus on time estimation?

6. Can we design one NN for one index?

7. Can we focus on more combination?

4.2 Resource Management System

4.2.1 Scheduling Scenario - Theory and Practice in Parallel Job Scheduling

4.2.2 Scalable system scheduling for HPC and big data

4.2.3 Poisson Distribution

The Poisson Distribution use a λ means the number of events happens between the average time

4.2.4 Insights in Hybrid Share

I. found an interesting insights in my past paper. The layout affects the interference of two applications. Balance and Continous. Maybe I can keep doing the research of this part.

2. neglect the impact of for one tasks, may have some master-slaver mode. The resources consumption is not homogeneous.

3. biparitate match is not the best. It cannot solve the problem of several jobs running together. It can olny deploy two jobs togther. It's worth mentioning that HybridShare neglect to place the multiple jobs in the job queues to the same nodes.

4. Can we use the algorithm into the cloud computing

4.2.5 异构计算系统调度理论与方法

1. 使用的计算资源具有多种类型的计算能力 SIMD MIMD 向量
2. 不同计算类型的计算资源能够相互协调运行
3. 不同子任务的并行性需求类型
4. 并行性 + 异构性

目标: 最短时间

4.2.6 Sinan: ML-Based and QoS-Aware Resource Management for Cloud Microservices

4.3 Application Framework Optimization

4.3.1 Graph System

Krill: A Compiler and Runtime System for Concurrent Graph Processing Notes:

1. Ligra

2.the time of loading graph nto memory consumes a lot.

4.3.2 Ray

1. Ownership: A Distributed Futures System for Fine-Grained Tasks [20] NSDI 2021;

- Introduction The original proposal uses synchronous calls that copy return values back to the caller.

  It suggest that , we need to pass the results to the master nodes after finish calculating. And in the end, the results will be passed to another nodes. There are too many such processes in the distribution computation. Thus, this process is wasting communication.

  Several recent systems [4, 34, 37, 45] have extended RPC so that, in addition to distributed communica- tion, the system may also manage data movement and paral- lelism on behalf of the application.

  4. PyTorch - Remote Reference Protocol.
  5. Ray v1.0. https://github.com/ray-project/ray/ releases/tag/ray-1.0.0.
  34. Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Eli- bol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerg- ing AI applications. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, 2018. USENIX Association.

37. Derek G. Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. CIEL: A universal execution engine for distributed data-flow computing. In Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI' 11, pages 113–126, Berke- ley, CA, USA, 2011. USENIX Association.

45. Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In Kathryn Huff and James Bergstra, editors, Proceedings of the 14th Python in Science Conference, pages 130 –136, 2015.

4.3.3 TurboTransformer

Jiazhi: NSCC-GZ Group Meeting 2021 - 09 - 30 Notes:

1. > 16 Threads Accelere not clearly
2. 32 64 performance degradation

3.thead only have a little time fully utilized CPU

4. tall-skinny

TSM2X: High-Performance Tall-and-Skinny Matrix-Matrix Multiplication on GPUs Cody Rivera, Jieyang Chen, Nan Xiong, Shuaiwen Leon Song, Dingwen Tao

4.3.4 Parameter Server

1. ParaX: Boosting Deep Learning for Big Data Analytics on Many-Core CPUs

- abstract: Insights: DL do not understand how to use CPU core. making many-core CPUs inef- ficient in DL computation. effectively alleviating bandwidth contention and CPU starvation.
  sufficiently overlaps the access-intensive layers with the compute- intensive ones to avoid contention, and proposes a NUMA-aware gradient server mechanism for training which leverages shared memory to substantially reduce the overhead of per-iteration parameter synchronization.

- FBLearner; Figure 1 shows that it is inefficient to assign only one instance to a many-core CPU which has much lower bandwidth. This is because one-instance-per-CPU implicitly imposes the per-layer barriers on the executions of the many cores which are jointly processing the batch.
- author find an interesting problem: some operations are bound in the memory, some operations are bound in compute.

2. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters

- Summation Service can be accelerated by AVX instructions and can be efficiently run on CPUs, while DNN model-related optimizer algorithms are run on GPUs for com- putation acceleration.

5 Computer Architecture

5.1 Roofline Model

5.2 Coursera: Computer Architecture, Computer Architecture: A Quantitative Approach

David Wentzlaff
Associate Professor
Department of Electrical Engineering
Princeton University

5.2.1 Readings: Pipelining: Basic and Intermediate Concepts

processor cycle: moving an instructions on step down the pipeline. Determined by the slowest stage.

5.2.2 Data Hazards

- **Schedule** explicitly avoids scheduling instructions that would create data hazards
- **Stall** freezes earlier stages until preceding instruction has finish producing data value
- **bypass** you can add extra hardware to your data path, and the extra hardware is going to send the value as soon as it gets created, so you may not have to wait for it to get to the end of the pipeline. So if the data value gets made early, you can just forward that to an instruction which needs it, but that adds extra hardware and complexity to your design.
- **Speculate** So, if you have a data hazard, you could assume it's not a problem or you could assume that, you know, everything's gonna be okay. I'll just use the encrypt value for a little bit of time and we'll assume that the value, the old value is equal to the new value or you do data speculations, other ways to do this. And if you make a mistake you catch it by the time you get to the end. And you basically have to re-execute the instruction with the correct value then.

References

[1] Li Yuan, Yunpeng Chen, Tao Wang, Weihao Yu, Yujun Shi, Zihang Jiang, Francis EH Tay, Jiashi Feng, and Shuicheng Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986*, 2021.

[2] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani,

Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[4] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560. Springer, 2020.

[5] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12239–12248, 2021.

[6] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[7] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer, 2020.

[8] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

[9] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[10] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the third ACM symposium on cloud computing*, pages 1–13, 2012.

[11] Nikita Mishra, John D Lafferty, and Henry Hoffmann. Esp: A machine learning approach to predicting application interference. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 125–134. IEEE, 2017.

[12] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Notices*, 48(4):77–88, 2013.

[13] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012.

[14] Jitendra Kumar and Ashutosh Kumar Singh. Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Generation Computer Systems*, 81:41–52, 2018.

[15] Weishan Zhang, Bo Li, Dehai Zhao, Faming Gong, and Qinghua Lu. Workload prediction for cloud cluster using a recurrent neural network. In *2016 International Conference on Identification, Information and Knowledge in the Internet of Things (IIKI)*, pages 104–109. IEEE, 2016.

[16] Vicent Sanz Marco, Ben Taylor, Barry Porter, and Zheng Wang. Improving spark application throughput via memory aware task co-location: A mixture of experts approach. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 95–108, 2017.

[17] Navaneeth Rameshan, Leandro Navarro, Enric Monte, and Vladimir Vlassov. Stay-away, protecting sensitive applications from performance interference. In *Proceedings of the 15th International Middleware Conference*, pages 301–312, 2014.

[18] Michael R Wyatt, Stephen Herbein, Todd Gamblin, Adam Moody, Dong H Ahn, and Michela Taufer. Prionn: Predicting runtime and io using neural networks. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–12, 2018.

[19] Josep Lluís Berral, Nicolas Poggi, David Carrera, Aaron Call, Rob Reinauer, and Daron Green. Aloja-ml: A framework for automating characterization and knowledge discovery in hadoop deployments. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1701–1710, 2015.

[20] Stephanie Wang, Eric Liang, Edward Oakes, Benjamin Hindman, Frank Sifei Luan, Audrey Cheng, and Ion Stoica. Ownership: A distributed futures system for fine-grained tasks. In *NSDI*, pages 671–686, 2021.