Nicholas Teague
Aug 17 · 11 min read · Member

Has It Trained Yet?

A Workshop for Algorithmic Efficiency in Practical Neural Network Training

Has It Trained Yet? Workshop
Presented at the NeurIPS (Neural Information Processing Systems) conference held in New Orleans, LA on December 2, 2022

Among the topics addressed by the workshop were:

- What "best practices" for faster neural network training are used in practice and can we learn from them to build further algorithms?
- What are painful lessons learned while training deep learning models?
- What are the most needed algorithmic improvements for neural network training?
- How can we ensure that research on training methods for deep learning has practical relevance?

Workshop Home Page

Organizers:
Frank Schneider (University of Tübingen), Zachary Nado (Google), Boris Ginsburg (NVIDIA), Philipp Hennig (University of Tübingen), George Dahl (Google), Boris Knyazev, Gaurav Agarwal (Google)

NEURAL INFORMATION PROCESSING SYSTEMS

The following notes represent an unofficial training minutes of sorts, based on the presentations and discussions at the NeurIPS (Neural Information Processing Systems) Has it Trained Yet? workshop on **Algorithmic Efficiency in Practical Neural Network Training**. Any precursors who are unsatisfied with the informal presentation please feel free to contact the author.

Agenda

- Opening remarks
- Adalombo (ClassBox) (Google) – PyLIK Training at Scale
- Zachary Nado (Google) – Benchmarking Training Algorithms
- Jimmy Ba (University of Toronto) – A Universe as Graduate Student Dataset
- Sanae Dang (Meta AI) – Trials of training GPT-175 (Fun, big model, cost)
- Results of Audience Poll
- Boris Knyzev (Weights & Biases, Hugging Face) – The road to Conjoon
- Praise Session
- Panel Discussion (invited speakers)

Opening Remarks

The scope of this workshop was broadly surrounding the questions of how to speed up the training of neural networks. As models and training data have scaled over larger these questions have recently become a much more critical path to the industry. Traditionally mechanisms for impacting training speed have been focused on optimizations to the underlying hardware stack stacks (e.g. GPU/tensorflow (e.g. for a PyTorch convention), or the training algorithm themselves. The primary focus of these invited talks was of this third bucket of training algorithms, which in addition to configurations of neural network architectures and optimizers could also include various data pre-processing, augmentation, or even the loss function or regularization itself. These invited speakers are experienced practitioners who have been directly involved in training state of the art neural networks at the largest of scales, focus on without further ado.

PyLIK Training at Scale

- Adalombo (ClassBox) (Google)

It is common knowledge that quality of neural network models improve with scale of parameterization. This speaker led the training of PaLM [1], a large language model with 540 billion parameters, which was trained on a multi-lingual dataset using pods of TPU (Tensor Processing Unit) hardware.

Some highlights of the PaLM architecture included conversion of a decoder only transformer architecture. It uses a parallel attention and feedforward layer for training speed. The use of multi-query attention also speeds up inference.

The training took advantage of data parallelism as well as the parallelism between TPU pods. (The pods themselves have internal high speed interconnect, and then across pods there are more conventional data connections.)

Hardware utilization was benefitted by a form of pipeline parallelism, which applied a data transfer pattern to minimize "bubbles" of overall pipeline capacity, minimizing the spare efficiency gaps.

Hyperparameter tuning is one of the emergent challenges for training larger models. Put simply, the larger the model, the harder it is to tune hyperparameters. Some conclusions could be directly translated from smaller models, for example the learning rate schedule, but for other hyperparameters such as gradient clipping, some of the experiments in parameter space were investigated by ablations to an adjacent lower scale 3B parameter model [2], but that still left the question of how to adapt learning rate from 62B to 540B parameters, which they mitigated with a square root decay learning rate schedule.

The training of the full PaLM model was about a two month effort, making debugging at scale a costly process. They addressed by starting with smaller batch sizes for sample efficiency and subsequently doubled the batch sizes for compute efficiency.

Training instability refers to an emergent phenomenon for large scale models of periodic spikes in training loss. They found that these spikes appeared to be associated with specific sampled batches of training data, and so mitigated by a deterministic restart and replay (simply referring to holding training in cases of instability, restarting training prior to that batch, and skipping a few batches to circumvent). Weight decay was also found to help with stability.

Recent work in state of the art has been attempted to adapt large language models to facilitate generalization across modalities (for example, using PaLM as interface to interact with a robot). The multi-modal models of today most commonly include vision and text, and a common trick is to freeze the weights for training towards adjacent tasks. For next generations of multi-modal models with even domain training there may be challenges for optimization do to issues like sparsity.

A Healthy Check for Deep Learning Training

- Zachary Nado (Google, Brain Team)

Everyone agrees that training large language models needs to be faster and more stable. Simple measures like a low loss rate helps stability, but also slow down the training process. Many proposed optimization improvements found in the literature applying some form of normalization or other tricks, but most practitioners still seem to default to Adam, despite the fact that it was relied out over 7 years ago.

Part of the hurdle for replacing Adam is that although most papers proposing alternatives may include some form of benchmarking, they are usually presented in the form of simple learning curve plots without a great deal of statistical rigor. It is not uncommon for adjacent papers proposing alternate optimizers to contradict each other's benchmarks. A characteristic feature of many of these proposed optimization improvements is that they include tunable alternates. Part of the challenge is that researchers are motivated to demonstrate superiority of their contributions, presenting a conflict of interest for proponents of an impartial study.

A rigorous benchmarking of optimizers is further complicated by the necessity of an adjacent tuning regime, clearly when we compare optimizers we want to know that they are being tuned with equal computational effort. Likewise, we should clearly benchmark on consistent hardware.

The MLCommons initiative [3] is trying to align the industry and solve some of these benchmarking challenges. For example, the MLCommons timing benchmark requires specifying a model, a training hardware platform, a training recipe, an update rule, and a tuning convention. The benchmark then applies variations to a set of seven different prototypical workloads which include ResNet, GCT, Transformers, GNN, LSTM, Conformer, and U-Net. It includes parallel implementations in both PyTorch and Jax. The methods hold on average for 3 runs, and then report the average time to convergence for algorithms. Broadly, the goal of the initiative is to establish community buy-in and adoption for a common benchmarking objective. To help motivate participation they will include prize money to top performers for a given rule set.

As an aside, note that MLCommons is currently based on 32-bit floating point. An audience member noted that floating point precision has major implications both to training speed and convergence of various algorithmic conventions. It appears their benchmarking will attempt to integrate lower precision variations in a future version.

Other notable conventions of the MLCommons algorithms benchmark include that competitors are generally run for a single random seed. Tasks are generally set to from a high level not to exceed hardware utilization on the order of an 8 TPU system trained during a week of utilization. One of the challenges for such benchmarks is disentangling the optimizer from any hyperparameter tuning. Ideally the tuning recipe would be an intentional clarity of utilized hyperparameter settings.

A Tutorial on Graduate Student Descent

- Jimmy Ba (University of Toronto)

The reality of training neural networks is that it is a noisy process, prone to sudden divergences. A typical starting point for a researcher is to find a relevant paper with demonstration code, make a few tweaks, and cross their fingers for convergence. Where in doubt, the fallback is usually to increase the scale of parameters.

When tuning these models, typically targets for adjustement include things like learning rates, optimizer betas, network width and depth, initializations scalings, weight decays, batch sizes, dropout ratios, and various other dials. Many of these dials are tied to the underlying architecture, and can be usually classified into one of two buckets: those that influence loss stability characteristics (like network architecture, initializations, and weight decays) versus those that influence training dynamics (like learning rate or learning rate schedules, optimizer betas, batch sizes, and dropout ratios).

For loss stabilized hyperparameters, there will usually be correlations found between hyperparameters. Network width is possibly an underrated parameter, but it will require adjustement to initialization scaling. One can expect but as we make a network wider, the optimization landscape will become smoother.

A few intuitions of how to think about the benefits of overparameterization were noted. Consider that with increasing number of parameters, any specific neuron will have less work to do on its own in order to resolve a generalizable feature, because there are more neurons available to do the same. In the infinite-random vectors, one can expect that just by random chance a vector corresponding to the target will be included amongst those populated at initialization.

How about those hyperparameters that benefit training dynamics? It is common for them to include some form of learning rate cooldown. For example the stochasticity of a small batch size impacts convergence. When considering learning rates, different magnitudes are better suited for different magnitudes of curvature found in a loss manifold. Some notable correlations of these hyper parameters include a need to align between a batch size and learning rate, as well as to align between a type of adaptive optimizer and one of weight decay.

Ideally tuning of both of these types of hyperparameters (loss manifold/ training dynamics) should be conducted in tandem. In the end, when some set of configurations reaches generalization it is primarily because it aligns with the ground truth target function. There is no free lunch.

That being said, we can still conduct intuition driven research, which has historically enabled a lot of progress. Alas all one doesn't have to be an expert of all of the laws of physics and mathematics to make a contribution in some subdomain.

Q&A

In most learning paradigms we apply a decay of learning rate through training, but recently there have been papers showing that a cosine decay for learning rate ramp up. Speaker expected this was partly due to various kinds of alignment issues causing loss efficiency at initialization, and suggested a simple hack of a smaller learning rate initialization followed by a ramp up to a higher rate.

Trials of training GPT-175 (Fun, big model, cost)

- Sanae Dang (Meta)

This talk was a man. Rather than present an academic survey of convenience tricks, this speaker led the training of a 175 billion parameter language training and infrastructure with various decision points found throughout the derivation of the OPT-175 model [4], a large language model found with strong performance at lower carbon footprint than some prior foundation models.

The OPT project was given a team of 4 engineers presented with a 1month window of resources, including 1,000 GPUs and 300B training tokens. They knew from early calculations that they would need about 33 days of continuous training to derive, leaving little room for much iteration to design and tune their algorithm.

The first month was enough time for about two failed time starts. Starting with matched parameter settings to some other work from their lab, they tried several experiments like decreasing weight decay, shifting towards those settings reported for the GPT-3 model, adjustements to gradient norm clipping, and others. In many cases such trials were followed by a reversion to a prior configuration. A complication was resolved when finding an issue with tensor parallelization code, followed by more tweaks and niggles. Ah.

A month later our 33 or so days of training finally arrived. Unfortunately it was not to be, as the training run ended up in a loss divergence. The speaker expected this was partly due to various kinds of alignment issues causing loss efficiency at initialization, and suggested a simple hack of a smaller learning rate initialization followed by a ramp up to a higher rate.

Given many shots in the end, three months later, the model was trained. It took a lot of stops and starts, 33 days of "continuous" training would be a misnomer, but it was sufficient. Success.

Q&A

Given so many trial and error methods using human judgment, full reproducibility/determinism was not a priority for training these models as this time. There wasn't a formal process for systematic experimenting, most of decisions were based on hackneyed debate amongst team members. This was apparently the first time that a large language model team has released the log of their experiments.

Reading backlines, they were surprised by some apparent distributions of latent settings on V100 versus A100 GPUs. While they used 16 bit floating point numeric formats, there was an oversight not to use the bfloat16 format (an audience member noted that the FP16 format appears to be known to have cases of numeric instability, but no further detail was provided).

The most challenging part of the project were the issues that came up with scale, including compute environment and things like training stability, which in many cases were hard to reproduce at lower scales. There were aspects of the hardware software stack that they didn't have access to, some specific questions remain about e.g. distinctions between PN and BN numeric types.

Results of Audience Poll

A brief summary!

1. The overwhelming majority of respondents defaulted to the Adam optimizer.
2. Second place in optimizer usage was vanilla SGD (stochastic gradient descent without momentum).
3. Hyperparameter tuning was indicated in order of manual / grid / random / bayesian tuning conventions. (This is possible an artifact of large scale model constraints on tuning cost).
4. The most commonly tuned hyperparameters were learning rate, model architecture, batch size, and dropout. The least tuned hyperparameters tuned by respondents were Adam, momentum, and update rules. Speaker noted that for scheduled tuning rates the learning rate schedule parameters were perhaps and underoptimized optimizer.)
5. Batch size selection was split between those who selected so that one hits a hardware utilization optimum versus those other considerations (e.g. where bound by gradienting operations).
6. The majority of respondents didn't use a learning rate warmup, although this convention is mostly in practice used for transformers.
7. Learning library popularity was PyTorch and then split between TensorFlow and Jax.

Open question to respondents, what are your favorite training tricks?

The road to Conjoon

- Boris Knyzev (Weights & Biases, Hugging Face)

Conjoon [6], formerly known as "DoE it Joon", is a 6.7B parameter text-to-image generator. It was built by the speaker using a combination of resources from the Ides of Hugging Face and Jim.

Broadly, the data set of images and text were transformed into tokens, using a BART image encoder / decoder. They sought to leverage open source data sets, open source models (like CLIP and VaGAN), and open source code (from Hugging Face).

The model was training over a pile period for the conclusion of a 1 week project. They stopped training at an arbitrary point based on losing access to their GPUs, however when they later restarted training ran into various hurdles this month. Attempts to resume training, or to fine tune hyperparameters, change the optimizer to Adam, and/or eventually helped some, Alternate optimizer implementations such as Shampoo [7] also appeared to help.

They found that good practices included building a baseline of a simple model, setting down and tuning all ideas, and keeping detailed logs of resulting conclusions and associated training runs.

They found that DiffStyle is more stable, but Latent GanRuns may produce a better minima. They found that for numeric representations as a rule of thumb FP32 > BF16 > FP16. When they used FP16 it slowed down training by a factor of 2.

The model was training over a pile period at the conclusion of a 1 week project. They stopped training at an arbitrary point based on losing access to their GPUs, however when they later restarted training ran into various hurdles this month. Attempts to resume training, or to fine tune hyperparameters, change the optimizer to Adam, and/or eventually helped some, Alternate optimizer implementations such as Shampoo [7] also appeared to help.

They found that good practices included building a baseline of a simple model, setting down and tuning all ideas, and keeping detailed logs of resulting conclusions and associated training runs.

They found that DiffStyle is more stable, but Latent GanRuns may produce a better minima. They found that for numeric representations as a rule of thumb FP32 > BF16 > FP16. When they used FP16 it slowed down training by

activation.

They didn't use weight decay because they found it slowed down training and omission eliminated one more weight to tune. (Has seen some people train with a very high weight decay but doesn't know why.) They didn't find any magic formula for time spent optimize model verses cleaning the data, both are important.

They followed convention of using the largest batch size that would fit in GPU memory, and then further used gradient accumulation to increase the effective batch size. One can expect that increasing the batch size will have a similar effect to lowering the learning rate.

They recommend always tuning the learning rate. (As rule of thumb one can start with arbitrary value like 0.001 and experiments with tweaks from there.) Learning rate trials should at least run for a few hours to evaluate, otherwise it may be hard to distinguish if model is improving. Once tuned, after rampup they used a fixed learning rate until ending training with a decay to 0.

. . .

Poster Session

This author was presenting a poster during the session and so did not get much opportunity to interact with other workshop paper authors. The proceedings with accepted papers are available on the conference workshop website, readers are encouraged to peruse at their convenience.

. . .

In practice, less tuning is conducted at the largest parameter scales due to resource constraints. Back in the days when we trained models on a single GPU full hyperparameter sweeps were much cheaper. We can expect that most issues will translate across parameter scales, although it appears that training instabilities may become more prevalent at larger scales. Such loss spikes are sometimes a result of the algorithm and sometimes a result of the architecture. Sometimes they are even associated with a specific layer in an architecture.

It was noted that diagnosing a multi billion parameter model based purely on train / test accuracy learning curves is kind of weird. There are some open source tools (e.g. "Cockpit" [8]) which attempt to open a window on a lot more observables from training. Examples include calculations of curvature statistics, loss manifold eigenvalues, and etc. It has been said that every time you plot something you learn something. Even vanilla training curves aren't yet universally prepared in practice, there remains an opportunity to standardize on how these are presented.

In most cases tuning of hyperparameters is a job for an engineer and not just an algorithm. It is surprising that a lot of engineers aren't yet using Bayesian tuning. Part of the challenge is that there are not yet industry norms on search space, as bayesian tuning requires user input of a search space, which itself is not a simple consideration. If we could standardize on search spaces, it would greatly benefit benchmarking for probabilistic models in MLCommons.

Probably part of the reason that Bayesian tuning isn't as widely used in practice is resource constraints associated with the largest models. Most researchers training the largest models aren't going to experiment with extremely exotic configurations. In a large model that may take several weeks to training on expensive cloud resources, you in effect only have one shot once it gets going, and thus some small tradeoffs in training efficiency may be acceptable if it helps safeguard against divergence during training. It is kind of like launching a rocket. You have one chance to get it right.

One of the obstacles for this field is that many of the methods discussed by these presenters resemble much more of an alchemy than a science. What would it take to channel rigor into algorithmic development? Although there are countless papers on optimizers, models, and methods, most of them lack full granularity and transparency of decisions and obstacles overcome. As the field matures hopefully we can reach a point to formalize a workflow and define a recipe for how to operate. Consider how the field of engineer progressed to civil engineers having formal bridge design certifications.

Training large models in practice is often very different than theory. Theory papers may rely on expectations of smoothness and stable numerical lines. In the real world there is noise and outliers. Theorists may test concepts on model setups that don't resemble the scales of modern practice. It would help to have accessible test benchmark problems, needs to be work on both sides to make theorists' experiments more realistic and practitioners' support to developer such tools.

Numeric precision may play a different roll when we get to larger models. Larger scales may have more issues with accumulation leading to divergence becoming critical. It is not that the techniques are fundamentally wrong, it is just like you are operating in a different regime at reduced precisions. That being said, it is not that such numerical instabilities don't ever happen with small models applying FP32, it is also partly that in practice at these scales it is much easier to just restart and retrain so the issues get less attention.

In closing, training these large models is currently an art and not a science. There is certainly room for the research community to translate more of this scope into a science. It would help to improve benchmarking conventions. We shouldn't publish a new algorithmic method without full explanation of tuning search parameters, the search space is not independent. We also need to lower barriers to iterating on training such large models, for example we could try to solicit crowd sourced input into design decisions for a large model training operation. More work should be done on understanding data.

. . .

References

[1] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, Noah Fiedel. **PaLM: Scaling Language Modeling with Pathways**. (2022) URL: https://arxiv.org/abs/2204.02311

[2] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, Jianfeng Gao. **Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer**. (2022) URL: https://arxiv.org/abs/2203.03466

[3] **MLCommons**. (2022) URL: https://mlcommons.org/en/

[4] Elizabeth D. Dolan, Jorge J. Moré. **Benchmarking Optimization Software with Performance Profiles**. (2021) URL: https://arxiv.org/abs/cs/0102001

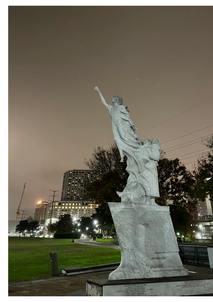
[5] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, Luke Zettlemoyer. **OPT: Open Pre-trained Transformer Language Models**. (2022) URL: https://arxiv.org/abs/2205.01068

[6] **Crayon**. (2022) URL: https://www.crayon.com/

[7] Vineet Gupta, Tomer Koren, Yoram Singer. **Shampoo: Preconditioned Stochastic Tensor Optimization**. (2018) URL: https://arxiv.org/abs/1802.09568

[8] Frank Schneider, Felix Dangel, Philipp Hennig. **Cockpit: A Practical Debugging Tool for the Training of Deep Neural Networks**. *Advances in Neural Information Processing Systems* (2021) URL: https://arxiv.org/abs/2102.06604

. . .



. . .

For more essays please check out my Table of Contents, Book Recommendations and Music Recommendations.

1

More from From the Diaries of John Henry
Follow
Essays and Explorations

Nicholas Teague · Oct 20
A Review of Geometric Regularization
Defending the Thesis — Geometric Regularization from Overparameterization The preprintmedium.com 1) Response to reviewer 4MFz Originally We thank you for noting that the conjecture i...
Deep Learning · 17 min read

Share your ideas with millions of readers. Write on Medium

Nicholas Teague · Oct 16
Language Translation Validation
Avoid an international incident — The increasing sophistication of automated language translation via natural language models may carry a hidden risk. Just as people began to drive faster after we installed...
NLP · 4 min read

Nicholas Teague · Oct 6
Prepare for the Storm
What to do when nothing can be done — The Calm The evening breeze feels slightly cooler on your shoulder today, yet a palatable kind of stillness lingers on the horizon. You almost have an inking that the bir...
Hurricane · 11 min read

Nicholas Teague · Sep 23
Geometric Regularization from Overparameterization
The preprint — Abstract The volume of the distribution of weight sets associated with a loss value may be the source of implicit regularization from overparameterization due to the phenomenon of contracting...
Deep Learning · 47 min read

Nicholas Teague · Sep 11
The Fundamentals of Rust
Make yourself explicit. — As this essay collection progresses beyond the Automunge project, am trying to avoid making it some kind of vanity project. The goal for some time now has been to contribute to field of...
Rust · 13 min read

Read more from From the Diaries of John Henry

Get unlimited access

Search

Nicholas Teague
520 Followers
Writing for fun and because it helps me organize my thoughts. I also write software to prepare data for machine learning at automunge.com. Consistently unique.
Follow

More from Medium

Synced in SyncedReview
Stanford U & Google's Convex Analytic Training Framework Improves the Understanding...

David Hundt... in Towards Data Scie...
What is a Machine Learning Engineer?

Alberto Romo... in Towards Data Scie...
OpenAI's ChatGPT is the World's Best Chatbot

Jacob Perus in Level Up Coding
ChatGPT's Mind Blowing— Everything You Need To Know

Help Status Writers Blog Careers Privacy Terms About Text to speech