Decentralized Deep Learning
Running Large Neural Networks Together

Max Ryabinin
Senior Research Scientist, Yandex
PhD Student, HSE University
Talk outline

› Motivation and key challenges

› Decentralized training
  • Specialized architectures
  • General data-parallel training
  • Pipeline-parallel training

› Decentralized inference of pretrained models
The state of deep learning in 2023

› Large-scale training becomes more popular

› Scaling laws promise continued gains

› Larger models have emergent abilities (e.g. in-context learning)

---

Arxiv.org/abs/2201.11990
Implications of scaling

〉 Some models cost hundreds of thousands to train — or more!

〉 GPT-3 training costs millions of $

〉 Hard to train for an average researcher and advance the field

img: jalammar.github.io/illustrated-bert
Solution: share the effort

› Collaboration has worked in other sciences

› Let's use idle volunteer resources for DL as well!
Challenges of volunteer deep learning

› **Node failures**: PC turns off, Internet gets disabled, ...

› **Communication over Internet**: magnitudes slower than clusters

› **Heterogeneous hardware**: different GPUs, connection speeds etc.
## Existing approaches for distributed DL

<table>
<thead>
<tr>
<th>Training method</th>
<th>Size limit</th>
<th>Throughput</th>
<th>Scalability</th>
<th>Fault tolerance</th>
<th>Worker hot-join</th>
<th>Network bandwidth</th>
<th>Network latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data parallel</td>
<td>Worker</td>
<td>High</td>
<td>Medium</td>
<td>Full</td>
<td>Yes</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Worker</td>
<td>High</td>
<td>High</td>
<td>Only workers</td>
<td>Yes</td>
<td>Medium</td>
<td>Any</td>
</tr>
<tr>
<td>Model parallel</td>
<td>System</td>
<td>Medium</td>
<td>Low</td>
<td>No</td>
<td>No</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Federated</td>
<td>Worker</td>
<td>Low</td>
<td>High</td>
<td>Only workers</td>
<td>Yes</td>
<td>Low</td>
<td>Any</td>
</tr>
</tbody>
</table>

**Desired**  
**System**  
High  
High  
Full  
Yes  
Low  
Any
Talk outline

› Motivation and key challenges

› Decentralized training
  • Specialized architectures
  • General data-parallel training
  • Pipeline-parallel training

› Decentralized inference of pretrained models
Learning@home (NeurIPS’20)
Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts

Workers
Neighbors
Experts
- selected
- others
Expert lookup
Inference
DMoE (Decentralized Mixture-of-Experts)
Training over the Internet

DHT request

Data transfer

- Trainer process
- Available expert (unused)
- Expert selected by gating function
- Failed expert (e.g. disconnected)
Moshpit SGD (NeurIPS’21)
Communication-Efficient Decentralized Training on Heterogeneous Unreliable Devices

› How to average the gating function/embeddings?

› We propose a new algorithm for decentralized AllReduce-like averaging

› Main idea: average in smaller non-overlapping groups

› Communication-efficient and fault-tolerant, useful even on its own
Moshpit All-Reduce: core idea
Experiments

![Graph showing top-1 validation accuracy over training time for different methods.](image1)

- AR-SGD, homog.
- AD-PSGD, homog.
- SGP, homog.
- Moshpit SGD, homog.
- AD-PSGD, heterog.
- SGP, heterog.
- Moshpit SGD, heterog.

![Graph showing training loss over time for different methods.](image2)

- AR-SGD, homog.
- Moshpit SGD, heterog.

$21-24$ (p3.24xlarge)

$15-16$

vast.ai+spot
Analysis TL;DR

The averaging converges exponentially quickly

**Theorem 3.2.** Consider a modification of Moshpit All-Reduce that works as follows: at each iteration $k \geq 1$, (1) peers are randomly split in $r$ disjoint groups of sizes $M_1^k, \ldots, M_r^k$ in such a way that $\sum_{i=1}^r M_i^k = N$ and $M_i^k \geq 1$ for all $i = 1, \ldots, r$ and (2) peers from each group compute their group average via All-Reduce. Let $\theta_1, \ldots, \theta_N$ be the input vectors of this procedure and $\theta_1^T, \ldots, \theta_N^T$ be the outputs after $T$ iterations. Also, let $\overline{\theta} = \frac{1}{N} \sum_{i=1}^N \theta_i$ Then,

$$E \left[ \frac{1}{N} \sum_{i=1}^N \| \theta_i^T - \overline{\theta} \|^2 \right] = \left( \frac{r-1}{N} + \frac{r}{N^2} \right)^T \frac{1}{N} \sum_{i=1}^N \| \theta_i - \overline{\theta} \|^2. \quad (5)$$

For Moshpit SGD — equivalent results to Local SGD

**Theorem 3.4** (Non-convex case). Let $f_1 = \ldots = f_N = f$, function $f$ be $L$-smooth and bounded from below by $f_\ast$, and Assumptions 3.1 and 3.2 hold with $\Delta_{pv}^k = \delta_{pv,1} \gamma E[\| \nabla f(\theta_k) \|^2] + L \gamma^2 \delta_{pv,2}^2$, $\delta_{pv,1} \in [0, 1/2)$, $\delta_{pv,2} \geq 0$. Then there exists such choice of $\gamma$ that $E \| \nabla f(\theta_{K \text{rand}}) \|^2 \leq \epsilon^2$ after $K$ iterations of Moshpit SGD, where $K$ equals

$$O \left( \frac{L \Delta_0}{(1-2\delta_{pv,2})^2 \epsilon^2} \left[ 1+ \tau \sqrt{1-2\delta_{pv,1} + \frac{\delta_{pv,2}^2 + \sigma^2/N_{\min}}{\epsilon^2} + \sqrt{(1-24\delta_{pv,1})(\delta_{pv,2}^2 + (\tau - 1)\sigma^2)} / \epsilon \right] \right),$$

$\Delta_0 = f(\theta^0) - f(\theta^\ast)$ and $\theta_{K \text{rand}}$ is chosen uniformly from $\{ \theta^0, \theta^1, \ldots, \theta^{K-1} \}$ defined in As. 3.2.

Again, if $\delta_{pv,1} \leq 1/3$, $N_{\min} = \Omega(N)$, $\delta_{pv,2} = \mathcal{O}(\sigma / N_{\min})$, and $\delta_{\eta} = \mathcal{O}((\tau - 1)\sigma)$, then the above theorem recovers the state-of-the-art results in the non-convex case for Local-SGD [64, 63].
DeDLOC (NeurIPS’21)
(Distributed Deep Learning in Open Collaborations)

› How to scale decentralized training to real-life scenarios?
› Propose an averaging algorithm that dynamically adapts to network conditions
› Recovers regular distributed methods in special cases
DeDLOC (NeurIPS’21)
(Distributed Deep Learning in Open Collaborations)

› For training, adopt large-batch SGD

› Accumulate batches on peers, synchronize when target size is reached

Learn more: huggingface.co/blog/collaborative-training
sahajBERT: the first collaboratively-trained LM

› We enlist the help of ~40 volunteers from the Bengali community

› Pretrain ALBERT-large, results competitive to SoTA trained on clusters

› Participants used their servers and even Colab/Kaggle instances!
What if some peers are malicious/faulty?

Can we train the model in such a way that no one peer can break it?

Hint: use cryptography :)
We develop a library for decentralized deep learning over the Internet.
Supports bypassing NAT, asynchronous training, data compression.
Data-parallel parts are tested in several practical projects.
Easy to use in standard PyTorch (just change ~2 lines of code!)
Integrated into PyTorch Lightning, used for Stable Diffusion finetuning.
How can we train large models over the Internet?

Key observation: with the growth in the hidden dimension size, compute costs grow faster than communication costs!

This observation can make training large models feasible for speeds <500Mb/s (especially if we compress activations)
SWARM Parallelism (ICML’23)

Training Large Models Can Be Surprisingly Communication-Efficient

We can use this fact for communication efficiency and create dynamically rebalanced pipelines for fault tolerance!
How to make it efficient?

› Server-side load balancing
  • If some servers disconnect, other servers close the gap

› Client-side routing
  • Clients choose servers with maximal throughput
Talk outline

〉 Motivation and key challenges

〉 Decentralized training

• Specialized architectures

• General data-parallel training

• Pipeline-parallel training

〉 Decentralized inference of pretrained models
Petals: Collaborative Inference and Fine-tuning of Large Models
(NeurIPS'22 “Broadening Research Collaborations” workshop, Best Paper Honorable Mention)

› We develop a system for running and fine-tuning LLMs over volunteer devices

› Instead of just getting model predictions, you can inspect its hidden states

› Possible to join the public swarm (serving BLOOM at the moment) or start your own

petals.ml
Many 100B+ language models were released

Meta AI is sharing OPT-175B, the first 175-billion-parameter language model to be made available to the broader AI research community.

Hard to use without multiple high-end accelerators!
Still, LLM.cuda() requires

8x NVIDIA RTX 3090 (24 GB) or 3x NVIDIA A100 (80 GB)

for 175B params in 8-bit
Option 1: Offloading

Load weights from RAM/disk on demand

Too slow for interactive inference
Option 2: Hosted APIs

- Hosted inference API

**Easy to use**

**Not flexible**

**Cost money**

```
Um "whatpu" é um pequeno animal peludo nativo da Tanzânia. Um exemplo de uma frase que usa a palavra whatpu é: Estávamos a viajar por África e vimos uns whatpus muito queridos. Fazer um "farduddle" significa saltar para cima e para baixo muito rápido. Um exemplo de uma frase que usa a palavra farduddle é:
```

- Sampling: greedy
  - Switch to "greedy" for more accurate completion e.g. math/history/translations (but which may be repetitive/less inventive)

  Compute
Existing solutions have limitations

Option 1. Offloading to RAM/SSD
- Inference is too slow for interactive apps
- 5.5 seconds/token in the fastest RAM offloading setup (needs 100+ GB RAM)
- 22 seconds/token in the fastest SSD offloading setup

Option 2. Hosted APIs
- No way to use custom fine-tuning and sampling methods
- No way to look at the block outputs and token probabilities
- Might be expensive
Our approach

Some participants (called **servers**) load BLOOM blocks to their GPUs and allow others to do forward and backward passes.
Our approach

- Some participants (called **servers**) load BLOOM blocks to their GPUs and allow others to do forward and backward passes.
- Other participants (called **clients**) perform forward/backward passes through the whole model by sending requests to servers.
Fast single-batch inference

**Offloading:**

\[ \approx 5-20 \text{ sec/token} \]

sends *hundreds of GiBs* over GPU bus

**Petals:**

\[ \approx 1 \text{ sec/token} \]

sends *MiBs* over the Internet
Each user can finetune the LLM for their own task

Low-rank adapters

Trainable prompts


import torch
from transformers import BloomTokenizerFast
from petals.client import DistributedBloomForCausalLM

MODEL_NAME = "bigscience/bloom-petals"
tokenizer = BloomTokenizerFast.from_pretrained(MODEL_NAME)
model = DistributedBloomForCausalLM.from_pretrained(MODEL_NAME)

inputs = tokenizer("A cat in French is \"", return_tensors="pt")["input_ids"]
remote_outputs = model.generate(inputs, max_new_tokens=3)
print(tokenizer.decode(remote_outputs[0]))
# Output: A cat in French is "chat",

tinyurl.com/petals-colab
Welcome! This chatbot runs BLOOMZ-176B over the Petals network. Please do not enter sensitive data and follow the model's terms of use. The chat history is recorded.

A human talks to a powerful AI that follows the human's instructions.

Human: Hi!

AI: Hi! How can I help you?

Human:
Conclusion

- Decentralized DL is a viable alternative to clusters
  - Open-source libraries allow easy adaptation from standard setups
  - Also useful for preemptible/spot instances (3x smaller cost)
- Ongoing challenges: data security, volunteer incentives, scheduling
- Curious to hear your thoughts on this line of research!
Other projects

Scaling Ensemble Distribution Distillation to Many Classes with Proxy Targets
(with Andrey Malinin, Mark Gales)

It's All in the Heads: Using Attention Heads as a Baseline for Cross-Lingual Transfer in Commonsense Reasoning
(with Alexey Tikhonov)

BLOOM: A 176B-Parameter Open-Access Multilingual Language Model
(as the Engineering and Scaling group chair at BigScience)
Thank you!

Max Ryabinin
Senior Research Scientist, Yandex
PhD Student, HSE University

✉️ mryabinin0@gmail.com
🔗 mryab.github.io