Lecture 28: The future of ML? Alternatives to the transformer. Open problems.

CS4787/5777 — Principles of Large-Scale Machine Learning Systems

Summary and open questions.

Scaling machine learning methods is increasingly important. In this course, we addressed the high-level question: **What principles underlie the methods that allow us to scale machine learning?** To answer this question, we used techniques from three broad areas: statistics, optimization, and systems. We articulated three broad principles, one in each area.

- **Statistics Principle: Make it easier to process a large dataset by processing a small random subsample instead.**
- **Optimization Principle: Write your learning task as an optimization problem, and solve it via fast general algorithms that update the model iteratively.**
- **Systems Principle: Use algorithms that fit your hardware, and use hardware that fits your algorithms.**

We covered many techniques in this class...**but there are lots of open questions left!**

▷ Open question: **Is scaling really all we need for ML**?

- Recent trend is to run bigger and bigger models!
- e.g. GPT-3 is a language model that has 175 billion parameters
- We can use these large models for zero-shot learning, and this often outperforms non-transfer-learning approaches
- Performance of these models seems to improve further with size, following so-called "scaling laws"
- Is scaling up the size of modern transformers where we should expect to see the most gains? Should we devote most of our resources to this?
- What is the right way to fine-tune foundation models for a target task?
  - Fine-tune all the weights
  - Fine-tune a prompt
  - Fine-tune some smaller subset of the weights
  - ...or learn a new network alongside the main one
- May be reaching the limits of scaling using language data, since an AI language model can only be as "smart" as a distribution over token strings

▷ Open question: **Can we find some new architecture to replace transformers**?

- Recall: transformers scale quadratically with context length
- This is bad for performance as we generate longer texts

- Alternatives:
  - State space models

$$x_k = Ax_{k-1} + Bu_k \qquad\qquad y_k = Cx_k.$$

  - Mixer architectures (long convolutional layers)
  - Models based on classes of matrices that admit fast matrix-vector multiply

▷ Open problem: **reproducibility and debugging of machine learning systems**.

- Most of the algorithms we discussed in class are randomized, and random algorithms are hard to reproduce.
- Even when we don't use explicitly randomized methods, floating point imprecision can still make results difficult to reproduce exactly.
  - For hardware efficiency, the compiler loves to reorder floating point operations (this is sometimes called fast math mode) which can introduce slight differences in the output of an ML system.
  - As a result, even running the same learning algorithm on the same data on different ML frameworks *can* result in different learned models!
- Reproducibility is also made more challenging when hyperparameter optimization is used.
  - Unless you have the random seed, it's impossible to reproduce someone else's random search.
  - Hyperparameter optimization provides lots of opportunity for (possibly unintentional) cheating, where the test set is used improperly.
- ML models are difficult to debug because they often **learn around bugs**.

▷ Open problem: **more scalable distributed machine learning**.

- Distributed machine learning has this fundamental tradeoff with the batch size.
  - Larger batch size good for systems because there's more parallelism.
  - Smaller batch size good for statistics because we can make more "progress" per gradient sample. (For the same reason that SGD is generally better than gradient descent.)
- Communication among workers outside a pod can have high latency in distributed learning.
- The datacenters of the future will likely have many heterogeneous workers available.
  - How can we best distribute a learning workload across heterogeneous workers?
- When running many workers in parallel, the performance will start to be bound by **stragglers**, workers that take longer to work than their counterparts. How can we deal with this while still retaining performance guarantees?

▷ Open problem: **robustness to adversarial examples**.

- It's easy to construct examples that fool a deep neural network.
- How can we make our scalable ML methods provably robust to these type of attacks?

Thank you, and please submit course evaluations!