Certigrad: Bug-Free Machine Learning On Stochastic Computation Graphs

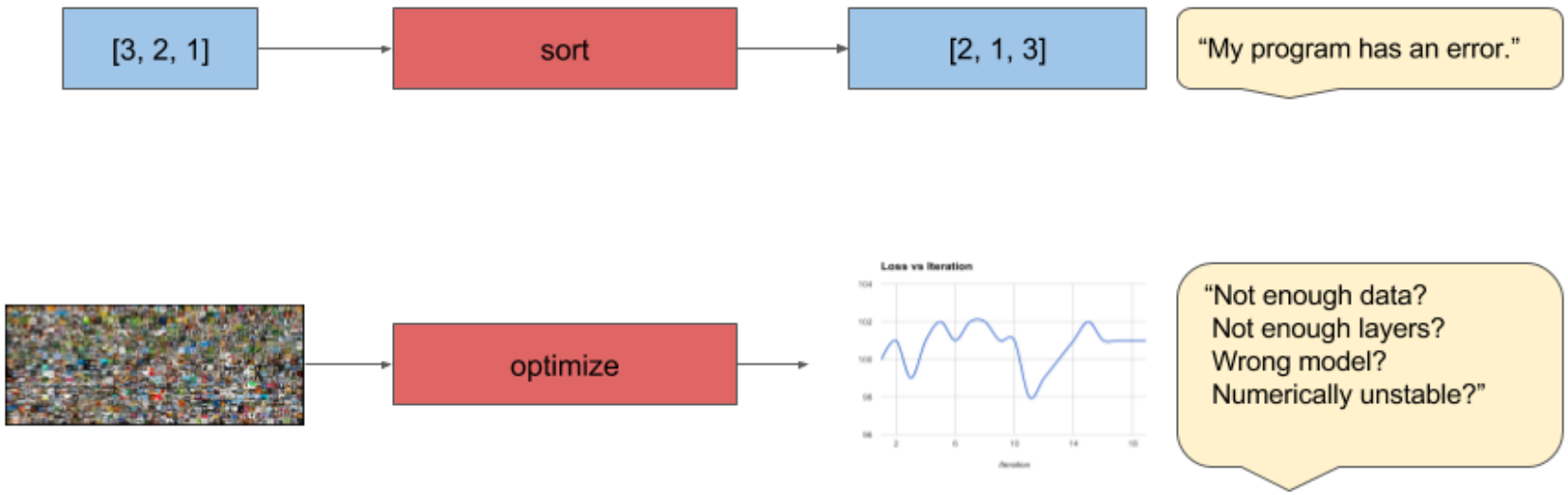
Daniel Selsam[1] Percy Liang[1] David Dill[1]

[1]Stanford University

July 20th, 2017

The Problem

It is extremely difficult to detect implementation errors in ML systems.



Other reasons to verify machine learning systems

- Tedious derivations can be automated.

$$\int_x \mathcal{N}(x; \mu, \sigma^2) \log \mathcal{N}(x; 0, I_n) = \cdots$$

$$= -\frac{1}{2}\left[\sum_{i=1}^{n} \left(\sigma_i^2 + \mu_i^2\right) + n \log 2\pi\right]$$

Other reasons to verify machine learning systems

- Tedious derivations can be automated.

$$\int_x \mathcal{N}(x; \mu, \sigma^2) \log \mathcal{N}(x; 0, I_n) = \cdots$$

$$= -\frac{1}{2}\left[\sum_{i=1}^{n} \left(\sigma_i^2 + \mu_i^2\right) + n \log 2\pi\right]$$

- Forgiving of inefficiencies (performance dominated by gemm).

Other reasons to verify machine learning systems

- Tedious derivations can be automated.

$$\int_x \mathcal{N}(x; \mu, \sigma^2) \log \mathcal{N}(x; 0, I_n) = \cdots$$

$$= -\frac{1}{2}\left[\sum_{i=1}^{n} \left(\sigma_i^2 + \mu_i^2\right) + n \log 2\pi\right]$$

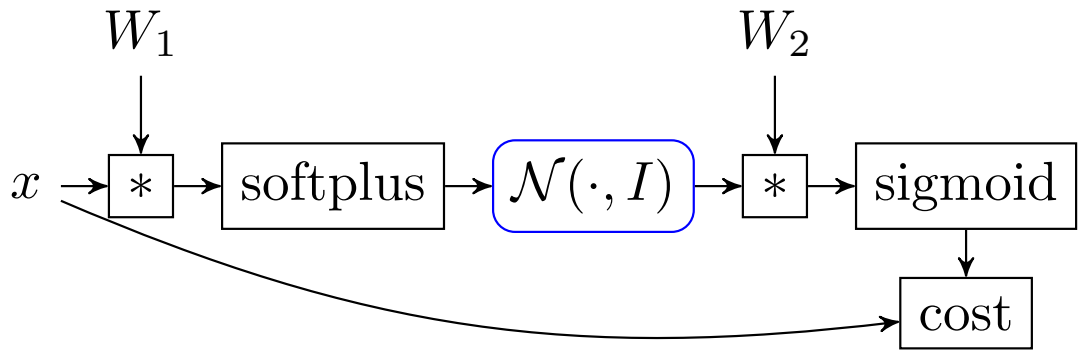
- Forgiving of inefficiencies (performance dominated by gemm).

- ML increasingly deployed in safety/security critical contexts.

Background: stochastic computation graphs

Stochastic computation graphs: non-recursive stochastic programs.

Example:



Loss function:

$$\mathcal{L}(W_1, W_2) = \mathbb{E}_{z \sim \mathcal{N}(\text{softplus}(W_1 x), I)} \left[\text{cost}(x, \sigma(W_2 z)) \right]$$

Goal: Given data x, find W_1, W_2 that minimize the cost.

Certigrad

- We built a system, *Certigrad* to optimize over stochastic computation graphs.

Certigrad

- We built a system, *Certigrad* to optimize over stochastic computation graphs.

- We proved it correct in Lean.

```
theorem backprop_correct {costs : list ID} :
  ∀ {nodes : list node} (inputs : env) (tgts : list reference),
  ∀ {tgt : reference} {idx : ℕ}, at_idx tgts idx tgt →
  nodup (tgts ++ map node.ref nodes) →
  well_formed_at costs nodes inputs tgt →
  grads_exist_at nodes inputs tgt →
  pdfs_exist_at nodes inputs →
  is_gintegrable (λ m, ⦀compute_grad_slow costs nodes m tgt⦀) inputs nodes dvec.head →
  can_differentiate_under_integrals costs nodes inputs tgt →

  ∇ (λ θ₀, E (graph.to_dist (λ m, ⦀sum_costs m costs⦀) (env.insert tgt θ₀ inputs) nodes) dvec.head) (env.get tgt inputs)
  =
  E (graph.to_dist (λ m, backprop costs nodes m tgts) inputs nodes) (λ dict, dvec.get tgt.2 dict idx) :=
```

Certigrad

- We built a system, *Certigrad* to optimize over stochastic computation graphs.

- We proved it correct in Lean.

```
theorem backprop_correct {costs : list ID} :
  ∀ {nodes : list node} (inputs : env) (tgts : list reference),
  ∀ {tgt : reference} {idx : ℕ}, at_idx tgts idx tgt →
  nodup (tgts ++ map node.ref nodes) →
  well_formed_at costs nodes inputs tgt →
  grads_exist_at nodes inputs tgt →
  pdfs_exist_at nodes inputs →
  is_gintegrable (λ m, ⦀compute_grad_slow costs nodes m tgt⦀) inputs nodes dvec.head →
  can_differentiate_under_integrals costs nodes inputs tgt →

  ∇ (λ θ₀, E (graph.to_dist (λ m, ⦀sum_costs m costs⦀) (env.insert tgt θ₀ inputs) nodes) dvec.head) (env.get tgt inputs)
  =
  E (graph.to_dist (λ m, backprop costs nodes m tgts) inputs nodes) (λ dict, dvec.get tgt.2 dict idx) :=
```

- We wrap Eigen for tensor operations
  - result: as fast as TensorFlow (on CPUs)

Resources

- Check out the code at: www.github.com/dselsam/certigrad.

- Read the paper at: https://arxiv.org/abs/1706.08605.