Orthonormality changes to Simplicits

Rishit Dagli

Contents

1 Motivation

When minimizing elastic energy functions, models often collapse to trivial solutions where all handles encode constant weights, effectively limiting the model to rigid motions.

$$\mathcal{L}_{\text{total}} = \lambda_1 \cdot \frac{\mathcal{E}_{\text{elastic}}}{N} + \lambda_2 \|\mathbf{W}^T\mathbf{W} - \mathbf{I}\|_F^2$$

where $\mathcal{E}_{\text{elastic}}$ is the elastic energy, N is the number of points, the second term enforces orthonormality of the outputs $\mathbf{W}$, and λ_1, λ_2 are Lagrangian multipliers.

Originally, in Simplicits $\lambda_1 = 10^{-1}$ and $\lambda_2 = 10^6$, which makes this way of enforcing these constraints seem like a hack to me. Thus, I was thinking of better ways to handle this.

2 What do I do?

2.1 Goals

Before stating the Algorithm, let us state our goals. We want to (1) at least get as good quality as the paper (we get better quality) and (2) run in times comparable to those listed in the supplementary (we run faster from my experiments!).

2.2 Main Idea

The primary idea is for 2D parameters, we apply a new recipe based on the quintic Newton-Schulz iteration to approximate orthogonalization:

$$\begin{aligned} \mathbf{X}_0 &= \frac{\mathbf{X}}{\|\mathbf{X}\| + \epsilon} \\ \mathbf{X}_{k+1} &= a\mathbf{X}_k + (b\mathbf{A}_k + c\mathbf{A}_k^2)\mathbf{X}_k \\ \text{where } \mathbf{A}_k &= \mathbf{X}_k\mathbf{X}_k^T \end{aligned}$$

with coefficients:

$$a = 3.4445, \quad b = -4.7750, \quad c = 2.0315$$

This iteration approximates the polar factor in the polar decomposition, effectively orthogonalizing the matrix while preserving its essential structure.

2.3 Algorithm

I present the algorithm for training in Algorithm 2. Alongside using this algorithm, I also remove the orthogonalization term from the loss completely.

Algorithm 1 A Modified Newton-Schulz Iteration.

Require: Matrix $\mathbf{X}$, Steps K

function ModifiedNS($\mathbf{X}$, K)
  Convert $\mathbf{X}$ to bfloat16 ▷ Reduce memory usage and improve speed
  $\mathbf{X} \leftarrow \frac{\mathbf{X}}{(\|\mathbf{X}\|+\epsilon)}$
  if rows $<$ columns **then** ▷ Work with tall matrices if they are not square
    $\mathbf{X} \leftarrow \mathbf{X}^T$
  end if
  for $k = 1$ to K **do**
    $\mathbf{A} \leftarrow \mathbf{X}\mathbf{X}^T$ ▷ Gram matrix
    $\mathbf{B} \leftarrow b\mathbf{A} + c\mathbf{A}^2$ ▷ Quintic parameters
    $\mathbf{X} \leftarrow a\mathbf{X} + \mathbf{B}\mathbf{X}$ ▷ Quintic iteration
    if $\mathbf{X}$ contains NaN or Inf **then**
      $\mathbf{X} \leftarrow \frac{\mathbf{X}}{\|\mathbf{X}\|}$
    end if
  end for
  if transposed in line 5 **then**
    $\mathbf{X} \leftarrow \mathbf{X}^T$
  end if
  Convert $\mathbf{X}$ to float32
  return $\mathbf{X}$
end function

Algorithm 2 Algorithm for Training, uses Algorithm 1.

Require: Parameters $\boldsymbol{\theta}$, Learning rate α, Momentum coefficients β_1, β_2

for all parameter $\mathbf{p}$ in $\boldsymbol{\theta}$ **do** ▷ Adam Updates
  Compute gradient $\mathbf{g}$
  $\mathbf{g} \leftarrow \text{clip}(\mathbf{g}, \text{max\_norm} = 1.0)$
  $\mathbf{m} \leftarrow \beta_1\mathbf{m} + (1-\beta_1)\mathbf{g}$
  $\mathbf{v} \leftarrow \beta_2\mathbf{v} + (1-\beta_2)\mathbf{g}^2$
  $\hat{\mathbf{m}} \leftarrow \mathbf{m}/(1-\beta_1^t)$
  $\hat{\mathbf{v}} \leftarrow \mathbf{v}/(1-\beta_2^t)$
  $\mathbf{p} \leftarrow \mathbf{p} - \alpha\hat{\mathbf{m}}/\sqrt{\hat{\mathbf{v}} + \epsilon}$
end for

for all 2D parameters **do**
  Partition parameters across GPUs
  for each GPU i **do**
    for all assigned parameter $\mathbf{p}$ **do**
      $\mathbf{p} \leftarrow$ ModifiedNS($\mathbf{p}$)
    end for
  end for
  Synchronize updates via all-reduce
  Apply synchronized updates
end for

3 Implementation Details

3.1 Parameter State Management

Each parameter maintains the following state:

1. Step count (t)
2. Exponential average of gradients ($\mathbf{m}$)
3. Exponential average of squared gradients ($\mathbf{v}$)

3.2 Distributed Processing

For N GPUs:

1. Each GPU processes approximately M/N parameters (where M is total 2D parameters)
2. Updates are accumulated in a flat buffer
3. All-reduce operation synchronizes updates across devices
4. Updates are redistributed to original parameter shapes

3.3 Memory Considerations

The optimizer requires the following additional memory per parameter:

- First moment buffer: Same size as parameter
- Second moment buffer: Same size as parameter
- Temporary buffers for Newton-Schulz: $\approx 3\times$ parameter size for 2D parameters