Orthogonality in Neural Networks

Weights and Gradients

Li Ju

Division of Scientific Computing
Department of Information Technology
Uppsala University

September 25, 2025

Orthogonality in Weights

The Problem

In neural networks, we typically work with unconstrained weight matrices, ignoring properties like orthogonality. However, this can lead to issues:

- Convolutional filters: highly correlated and redundant.
- Linear weights: long tailed spectrum.
- Signals and gradients: amplified or diminished as they pass through networks.

The Solution: Orthogonality

Enforcing orthogonality in weight matrices can mitigate these problems.

Advantages of Orthogonal Weights

- **Gradient Stability:**
  - Orthogonal layers are norm-preserving, which prevents gradients/signals from exploding or vanishing..
- **Signal Preservation:**
  - In Recurrent Neural Networks (RNNs), preserving the signal norm is critical for learning long-term dependencies.
- **Reduced Redundancy:**
  - Promotes diverse and non-correlated features in convolutional layers.

Three Main Approaches

1. **Regularization**
   - Add a penalty term to the loss function to encourage orthogonality.
2. **Optimization on Manifolds**
   - Treat the set of orthogonal matrices as a manifold and perform constrained optimization.
3. **Parameterization**
   - Orthogonality by construction, using specific parameterizations that ensure orthogonality.

This talk is intended as a brief survey of these methods, giving you a collection of pointers for these techniques to explore further if you are interested.

Regularization

We can add a regularizer to the main loss function to encourage the weight matrix $W \in \mathbb{R}^{m \times n}$ to be orthogonal.

Soft Orthogonality (SO) Regularizer

$$\mathrm{SO}(W) := \lambda \| I_n - W^\top W \|_F$$

Double-Sided Orthogonality (DSO) Regularizer

$$\mathrm{DSO}(W) := \lambda \left(\| I_n - W^\top W \|_F + \| I_m - W W^\top \|_F \right)$$

- This approach is simple to implement and often serves as a direct replacement for ℓ_2 regularization.

Optimization

This approach frames the problem as a constrained optimization task.

$$\arg\min_{W \in \mathbb{R}^{m \times n}} R(W) \quad \text{s.t.} \quad W^\top W = I_n$$

The constraint set defines the **Stiefel Manifold**, $V_n(\mathbb{R}^m)$:

$$V_n(\mathbb{R}^m) := \{Y \in \mathbb{R}^{m \times n} : Y^\top Y = I_n\},$$

with a degree of freedom $mn - \frac{n(n+1)}{2}$ ($n(n-1)/2$ for square matrices).

To solve this, we use methods like **Riemannian SGD**, which adapts standard gradient descent to operate on the geometry of the manifold.

Optimization with Riemannian SGD

Standard gradient steps would move the weights off the manifold. Riemannian SGD corrects this.

Each iteration involves four key steps:

1. Compute ambient gradient in Euclidean space: $\nabla_A R(W^t)$.
2. Project it onto the tangent space, to get the Riemannian gradient: $\nabla_T R(W^t)$.
3. Move along the descent direction constructed from the Riemannian gradient.
4. "Retract" the resulting point back onto the manifold.

For Stiefel manifolds, the calculation of Riemannian projection, construction of descent directions and retraction can be analytically performed. For more details, check[1].

[1] Tagare, "Notes on optimization on stiefel manifolds".

Parameterization

For other approaches, we have $y = \mathrm{act}(x^\top W + b)$ where $\theta = W$ are the learnable parameters.

The Core Idea of Parameterization

Instead of learning $\theta = W$ directly, we define W as a differentiable function of some underlying parameters θ:

$$W = \mathrm{OMC}(\theta) \quad \text{such that} \quad W^\top W = I$$

where OMC is an Orthogonal Matrix Constructor.

- The network layer becomes $y = \mathrm{act}(x^\top \mathrm{OMC}(\theta) + b)$.
- We can then learn θ using any standard optimizer (e.g., Adam, SGD), eliminating the need for manifold methods or regularization terms.

Parameterization Methods

Orthogonal matrices can be constructed from simpler building blocks.

Householder Reflections

$$R = I - 2\frac{vv^H}{\|v\|^2}$$

Any $n \times n$ unitary matrix W can be decomposed into a product of n Householder reflections:

$$W = \prod_{i=1}^{n} R_i \quad \text{where} \quad R_i = I - 2\frac{v_i v_i^H}{\|v_i\|^2}$$

Here, $\theta = \{v_i\}_i^n$ are the learned parameters.

Givens Rotations

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & \cos\theta & \cdots & -\sin\theta & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & \sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

where $G(i,j,\theta)$ is an identity matrix with the $(i,i),(j,j),(i,j),(j,i)$ entries replaced.

Any $n \times n$ orthogonal matrix W is a product of $n(n-1)/2$ Givens rotation matrices:

$$W = \prod_{i,j\in[n],i<j} G(i,j,\theta_{i,j})$$

Here, $\theta = \{\theta_{i,j}\}_{i,j\in[n],i<j}$ are the learned parameters.

Cayley Transform

An efficient parameterization for the full set of orthogonal matrices:

$$W = (I + A)^{-1}(I - A)D$$

Here, A is a skew-symmetric matrix whose $\frac{n(n-1)}{2}$ upper-triangular entries are the learned parameters. D is a diagonal matrix of ± 1.

Implementation

PyTorch provides a simple way to apply this via `torch.nn.utils.parametrizations.orthogonal`.

Orthogonality in Gradients

Orthogonality in Gradients

Muon, a recent optimizer, has achieved

- current training speed records for both NanoGPT and CIFAR-10 speedrunning.
- $\sim 30\%$ reduced computation cost over AdamW on LLM training.

The Muon Algorithm

1. Compute gradient: $G_t = \nabla R(W_t)$.
2. Update momentum: $M_t = \mu M_{t-1} + G_t$.
3. **Orthogonalize direction:** $O_t = \text{NewtonSchulz}(M_t)$.
4. Update parameters: $W_{t+1} = W_t - \eta O_t$.

What is the Orthogonalization Step?

The key step is the Newton-Schulz iteration, which approximates the orthogonal component of the momentum matrix.

- Consider the Singular Value Decomposition (SVD) of the momentum:

$$M_t = U_t \Sigma_t V_t^\top$$

- The Newton-Schulz step computes an orthogonal matrix O_t that approximates:

$$O_t \approx U_t V_t^\top$$

- In essence, it takes the update direction M_t and **discards its singular values**, keeping only the rotational part.

Why Does This Work? Controlling Output Change

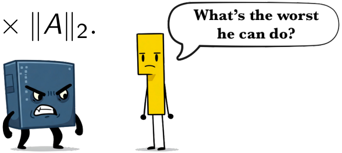
The goal is to update weights to decrease the loss, while controlling both input and output vectors to be "informatively dense" (i.e. $\|\cdot\|_{\text{RMS}} \approx 1$).

Consider a linear layer with weights W and input x:

- The change in output is $\Delta y = \Delta W x$, where we want $\|\Delta y\|_{\text{RMS}} \downarrow$.
- We can measure this change using the Root-Mean-Square (RMS) norm. The relevant operator norm is:

$$\|A\|_{\text{RMS}\to\text{RMS}} = \max \frac{\|Av\|_{\text{RMS}}}{\|v\|_{\text{RMS}}} = \sqrt{\frac{d_{\text{in}}}{d_{\text{out}}}} \times \|A\|_2.$$

$$\|\Delta y\|_{\text{RMS}} \downarrow \iff \|\Delta W\|_{\text{RMS}\to\text{RMS}} \downarrow \iff \|\Delta W\|_2 \downarrow$$



Why Does This Work? Controlling Output Change

The Constrained Optimization Problem

Muon implicitly solves: "Find the update ΔW that maximally decreases the loss for a fixed-size change in the output."

$$\min_{\Delta W} \langle R(W), \Delta W \rangle \quad \text{s.t.} \quad \|\Delta W\|_2 \leq \eta$$

The solution to this problem is $\Delta W^\star \propto UV^\top$, where $\nabla R(W) = U\Sigma V^\top$. This is exactly what Muon computes.

Thank you for listening!

Questions?