orthogonality_in_nn

Orthogonality is one of the most important concepts in linear algebra. However in neural network, we almost exclusively work with unconstrained matrices, not caring about orthogonality no matter for weights, for embedding/output or for gradients. Recently I happen to read a bit more about a few loosely connected papers in this field and here are the summaries. Note that all the discussions here can be naturally extended to the complex space for unitary matrices.

Why we may want orthogonality in weights

Here I list the advantages of both orthogonalized weight initialization (key word: dynamical isometry) and weight orthogonalization, where are slightly different but closely connected.

- Convolutional filters in deep neural network are high correlated (redundant).
- Each convolutional filter/linear layers as a linear operator has long-tailed spectrum [1].
- Unlike Lipschitz-constrained layers, orthogonal layers are energy preserving, avoiding amplifying or diminishing signals and gradient-norm-preserving, preventing gradient explosion/vanishing, which enables training very deep neural network without skip connections nor batch normalization [2].
- In RNN, signal-preserving is valued for remembering long-term dependencies, thus quite some works have been done there [3].

How to encourage orthogonality then?

There are generally two approaches for orthogonalization of weights: regularization, parameterization and optimization.

Regularization

Let us consider a linear layer $y = \mathrm{act}(Wx + b)$, where we want to encourage $W \in \mathbb{R}^{m\times n}$ to be orthogonal. The baseline regularizer is defined as Soft Orthogonality (SO) regularizer:

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

where I_n denote identity matrix of dimension n and $\|\cdot\|$ denotes some kinds of norm, e.g., spectral norm $\|\cdot\|_2$, Frobenius norm $\|\cdot\|_F$. Alternatively, there is also

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

which is equivalent to $\mathrm{SO}(W)$ when $m = n$.

For convolutional layers, there been quite some works to design better orthogonality regularizers [4, 5].

Practically, it is recommended to replace ℓ_2 regularizer with weight orthogonality regularizer.

Optimization

It is easy to initialize weight matrices to be orthogonal. Then during the optimization process, if we can manage to keep the updated weights of each step be orthogonal via constrained/manifold optimization, we can also ensure the final weights to be orthogonal. Formally, we are aiming to solve the following optimization problem:

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

The constraint set is defined as Stiefel manifold of orthogonal n-frame in $\mathbb{R}^m$, i.e.,

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

with a dimension of $mn - \frac{n(n+1)}{2}$.

To solve the problem, we can use Riemannian SGD [6] (the most formal way):

1. Compute the ambient gradient at W^t as $\nabla_A R(W^t)$.
2. Compute Riemannian gradient $\nabla_T R(W^t)$.
3. Construct a search curve
4. Step forward with step size η.

From step 2, alternatively one can:
3. Move along the gradient in the tangent space $W_T^{t+1} = W^t - \eta \nabla_T R(W^t)$. Notice, this generally gives a point off the manifold.
4. Retract/project the point back to the manifold $W^{t+1} = \text{ret/proj}(W_T^{t+1})$.

The cheapest way: One can also directly move following the ambient gradient and retract/project the new point back (general projected gradient descent), which is cheaper but with no convergence guarantee.

Gradient in the tangent space

For Stiefel manifold, the tangent space at W is given by

$$T_W V_n(R^m) = \{Z^\top W + W^\top Z = 0 : Z \in \mathbb{R}^{m \times n}, W \in V_n(R^m)\}.$$

Thus, $Z^\top W$ is a skew symmetric matrix.

We denote the ambient gradient as $G = \nabla_A R(W)$. The tangent gradient is given by

$$\nabla_T R(W) = \underbrace{\left(GW^\top - WG^\top\right)}_{A} W.$$

Note that A is a skew symmetric matrix. One can verify that $\nabla_T R(W)$ is in the tangent space:

$$\begin{aligned} W^\top A^\top W + W^\top A W &= W^\top (W G^\top - G W^\top) W + W^\top (G W^\top - W G^\top) W \\ &= G^\top W - W^\top G + W^\top G - G^\top W \\ &= 0. \end{aligned}$$

Constructing search direction

The search direction can be constructed with Cayley transformation.
Let $W \in V_n(R^m)$ and S by any matrix that $S^\top = -S$ (skew symmetric), Cayley transformation is defined as

$$Y(\tau) = \left(I + \frac{\tau}{2} S\right)^{-1} \left(I - \frac{\tau}{2} S\right) W.$$

We notice that

1. It stays strictly in the Stiefel manifold
2. Its tangent derivative at $\tau = 0$ is $Y'(0) = -SW$.

If we choose $S = A$, we have $Y'(0) = -AW = -\nabla_T R(W)$, i.e., it is a descent direciton/curve and taking a small enough step size η ensures descent.

Note:

1. For other technical details you can refer to [6], like how to efficiently compute the inverse of $I + \frac{\tau}{2} A$.
2. For other adaptive methods on Riemannian manifold, one can refer to [7]

Parameterization

In general for a linear layer $y = \text{act}(W(\theta)x + b)$, where $W = \theta$, i.e., each elements of the weight matrix is a single parameter. With this parameterization, it is hard to ensure orthogonality of W by nature.
Alternatively, we can construct use other parameterization in the following form:

$$y = \text{act}(W(\theta) + b),\ W = \text{OMC}(\theta).$$

If the orthogonal matrix constructors (OMC) are differentiable, we can get rid of the manifold-constrained optimizers and soft penalty. Following are a few commonly used methods for orthogonal matrix construction. Here we slightly expand our scope to unitary matrices ($AA^H = I$ where A^H denotes the conjugate transpose).

Building blocks

Householder reflection

For any nonzero vector $v \in \mathbb{C}^n$, its householder matrix is

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

Here R hermitian $R = R^H$ and unitary $RR^H = I$.

Theorem: All $n \times n$ unitary matrix can be decomposed into a product of n Householder matrices $R_1, \ldots, R_n$ [8]:

$$A = \prod_{i=1}^{n} R_i, \quad R_i = I - 2\frac{v_i v^H}{\|v_i\|^2}, \quad v_i \in \mathbb{C}^n.$$

Givens rotation

First initialize a Givens block

$$G = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}.$$

To construct an $n \times n$ orthogonal matrix, embed G into I_n at row i and column j. For example, with $n = 5$,

$$G(2,4,\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & \sin\theta & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

which is orthogonal.

Theorem: Any orthogonal matrix $W \in O(n)$ can be written as a product of $n(n-1)/2$ real Givens rotations

$$W = \prod_{p,q} G(p, q, \theta_{p,q}).$$

With matrix multiplication, one can use $n(n-1)/2$ parameters to span the full set of orthogonal matrices.

Diagonal phase matrix + (inverse) Fourier transform

For any diagonal matrix D with $D_{j,j} = e^{i\theta_j}$ with parameters $\theta_j \in \mathbb{R}$, D is Hermitian and unitary. Also, both $\mathcal{F}$ and $\mathcal{F}^{-1}$ are Hermitian and unitary. Combining D and $\mathcal{F}/\mathcal{F}^{-1}$ gives unitary dense matrix.

By adding these building blocks together, one can obtain various parameterization methods to construct unitary matrices [9, 10].

A more efficient way: Cayley transformation

Theorem: Every orthogonal matrix W can be expressed as

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

where $A = \{a_{i,j}\}$ is real-valued skew-symmetric with $|a_{i,j}| \leq 1$ and D is diagonal with all nonzero entries equal to ± 1.

With a predefined D (acting as a hyperparameter), we can parameterize orthogonal matrices with A using $\frac{n(n-1)}{2}$ parameters, spanning the full set of orthogonal matrices.

Efficient gradient computation: Refer to the original paper [11].

Implementation

Pytorch supports orthogonal weights by parameterization. You can read here.

Orthogonality in Gradient

Recently, a new optimizer Muon [12] is proposed and has outperformed AdamW in a lot of benchmark tasks, reducing the training cost by 1/3 on average. The optimizer is designed based on orthogonalized gradient. Instead of treating weights as vectors, they treat weights as matrices (linear operators).

It is performed as follows:
Initialize $M_0 = 0$

1. Compute the 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 $W_t = W - \eta O_t$.

Everything is pretty standard except the Newton Schulz step. What does it do? For raw direction M_t, we always have

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

Newton Shultz step is to approximate the orthogonalized direction: $\text{NewtonShulz}(M_t) \approx U_t V_t^\top$, removing the singular values.

Why this is helpful for optimization? For linear layers, we are mapping dense input to dense output. If we want to keep the training stable, we need to keep norm of data flow stable too. We mean "stable" by the average size of entries of input/output are around one, i.e., $\|v\|_{\text{RMS}} = \sqrt{\sum_{i=1}^d v_i^2/d} = \sqrt{1/d}\|v\|_2$.

But how the linear layers / matrix weights affect the RMS norm of the data? It is defined by the operator norm of a matrix:

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

For parameter updates $W + \Delta W$, the resulting output change is given by

$$\Delta y = (W + \Delta W)x - Wx = \Delta W x.$$

Further we have

$$\|\Delta y\|_{\text{RMS}} = \|\Delta W x\|_{\text{RMS}} \leq \|\Delta W\|_{\text{RMS}\to\text{RMS}} \times \|x\|_{\text{RMS}}.$$

So we want to minimizing the loss function while maintaining a bound on the amount of the output change (in RMS norm). So, we are interested in the optimization problem

$$\min_{\Delta W} < \nabla R(W), \Delta W > \quad \text{s.t.} \quad \|\Delta y\|_{\text{RMS}} \leq \eta.$$

If we have $\|x\|_{\text{RMS}} = 1$, equivalently we have

$$\min_{\Delta W} < \nabla R(W), \Delta W > \quad \text{s.t.} \quad \|\Delta W\|_{\text{RMS}\to\text{RMS}} \leq \eta.$$

Then given that $\nabla R(W) = U\Sigma V^T$, we have $\Delta W^\star = -\eta \times \sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} \times UV^\top$.

References

[1] Wang, Jiayun, et al. "Orthogonal convolutional neural networks." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020.
[2] Xiao, Lechao, et al. "Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks." *International conference on machine learning*. PMLR, 2018.
[3] Helfrich, Kyle, Devin Willmott, and Qiang Ye. "Orthogonal recurrent neural networks with scaled Cayley transform." *International Conference on Machine Learning*. PMLR, 2018.
[4] Xie, Di, Jiang Xiong, and Shiliang Pu. "All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.
[5] Bansal, Nitin, Xiaohan Chen, and Zhangyang Wang. "Can we gain more from orthogonality regularizations in training deep networks?." *Advances in Neural Information Processing Systems* 31 (2018).
[6] Tagare, Hemant D. "Notes on optimization on stiefel manifolds." *Yale University, New Haven* (2011).
[7] Becigneul, Gary, and Octavian-Eugen Ganea. "Riemannian Adaptive Optimization Methods." *International Conference on Learning Representations*.
[8] Uhlig, F. Constructive Ways for Generating (Generalized) Real Orthogonal Matrices as Products of (Generalized) Symmetries. Linear Algebra and its Applications, 2001.
[9] Arjovsky, Martin, Amar Shah, and Yoshua Bengio. "Unitary evolution recurrent neural networks." *International conference on machine learning*. PMLR, 2016.
[10] Mhammedi, Zakaria, et al. "Efficient orthogonal parametrisation of recurrent neural networks using

householder reflections." *International Conference on Machine Learning*. PMLR, 2017.
[11] Helfrich, Kyle, Devin Willmott, and Qiang Ye. "Orthogonal recurrent neural networks with scaled Cayley transform." *International Conference on Machine Learning*. PMLR, 2018.
[12] https://kellerjordan.github.io/posts/muon/