Kronecker Recurrent Units: Supplementary Material

Cijo Jose [1 2] **Moustapha Cissé** [3] **François Fleuret** [1 2]

1. Analysis of Vanishing and Exploding Gradients in RNN

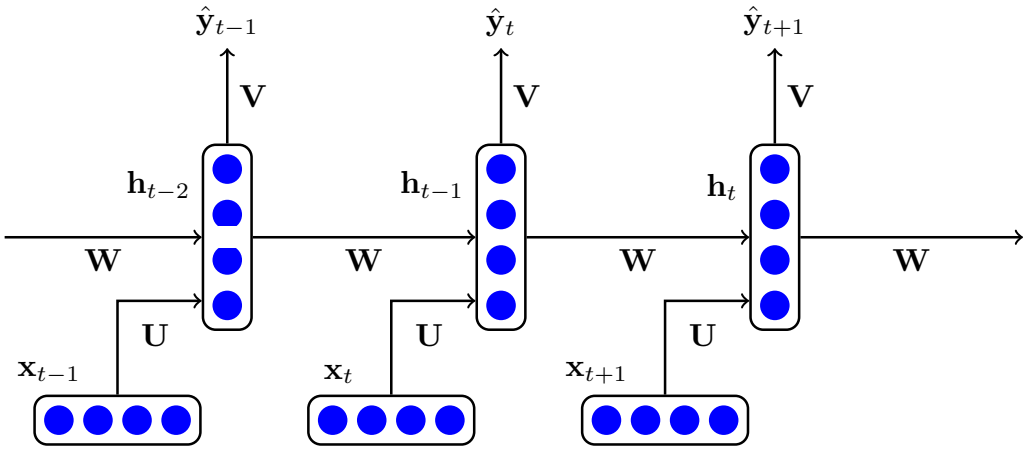

Figure 1. A slice of a recurrent neural network unrolled along the time. RNN is a very deep network along the time with shared parameters. At time step t RNN receives the input $\mathbf{x}_t$ and this input is encoded using the input to hidden matrix $\mathbf{U}$. This encoded input is combined with the information from the previous hidden state $\mathbf{h}_{t-1}$ using the recurrent weight matrix $\mathbf{w}$ to get the next hidden state. The information from the hidden state is extracted using the hidden to output matrix $\mathbf{V}$ also called as decoder to predict the targets $\hat{\mathbf{y}}_t$.

Figure 1 illustrates a recurrent neural network unrolled along time. Given a sequence of T input vectors: $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1}$, let us consider the operation at the hidden layer t of a recurrent neural network:

$$\mathbf{z}_t = \mathbf{W}_t \mathbf{h}_{t-1} + \mathbf{U}_t \mathbf{x}_t + \mathbf{b} \tag{1}$$

$$\mathbf{h}_t = \sigma(\mathbf{z}_t) \tag{2}$$

By the chain rule,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \tag{3}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \prod_{k=t}^{T-1} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \prod_{k=t}^{T-1} \mathbf{J}_{k+1} \mathbf{W}^T \tag{4}$$

[1]Idiap Research Institute [2]École Polytechnique Fédérale de Lausanne (EPFL) [3]Facebook AI Research. Correspondence to: Cijo Jose <cijo.jose@idiap.ch>, Moustapha Cissé <moustaphacisse@fb.com>, François Fleuret <francois.fleuret@idiap.ch>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

where σ is the non-linear activation function and $\mathbf{J}_{k+1} = diag(\sigma^{'}(\mathbf{z}_{k+1}))$ is the Jacobian matrix of the non-linear activation function.

$$\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \right\| = \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \prod_{k=t}^{T-1} \mathbf{J}_{k+1} \mathbf{W}^T \right\| \tag{5}$$

$$\leq \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \right\| \prod_{k=t}^{T-1} \left\| \mathbf{J}_{k+1} \mathbf{W}^T \right\| \tag{6}$$

$$\leq \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \right\| \left\| \mathbf{W} \right\|^{T-t} \prod_{k=t}^{T-1} \left\| \mathbf{J}_{k+1} \right\| \tag{7}$$

From equation 7 it is clear the norm of the gradient is exponentially dependent upon two factors along the time horizon:

- The norm of the Jacobian matrix of the non-linear activation function $\|\mathbf{J}_{k+1}\|$.
- The norm of the hidden to hidden weight matrix $\|\mathbf{W}\|$.

These two factors are causing the vanishing and exploding gradient problem.

Since the gradient of the standard non-linear activation functions such as tanh and ReLU are bounded between [0, 1], $\|\mathbf{J}_{k+1}\|$ does not contribute to the exploding gradient problem but it can still cause vanishing gradient problem.

2. Long Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997)

LSTM networks presented an elegant solution to the vanishing and exploding gradients through the introduction of gating mechanism. Apart from the standard hidden state in RNN, LSTM introduced one more state called cell state c_t. LSTM has three different gates whose functionality is described as follows:

- Forget gate ($\mathbf{W}_f, \mathbf{U}_f, \mathbf{b}_f$):Decides what information to keep and erase from the previous cell state.
- Input gate ($\mathbf{W}_i, \mathbf{U}_f, \mathbf{b}_i$): Decides what new information should be added to the cell state.
- Output gate ($\mathbf{W}_o, \mathbf{U}_o, \mathbf{b}_o$):Decides which information from the cell state is going to the output.

In addition to the gates, LSTM prepares candidates for the information from the input gate that might get added to the cell state through the action of input gate. Let's denote the parameters describing the function that prepares this candidate information as $\mathbf{W}_c, \mathbf{U}_c, \mathbf{b}_c$.

Given a sequence of T input vectors: $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1}$, at a time step t LSTM performs the following:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{x}_t + \mathbf{b}_f) \tag{8}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{U}_i \mathbf{x}_t + \mathbf{b}_i) \tag{9}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{x}_t + \mathbf{b}_o) \tag{10}$$

$$\hat{\mathbf{c}}_t = \tau(\mathbf{W}_c \mathbf{h}_{t-1} + \mathbf{U}_c \mathbf{x}_t + \mathbf{b}_c) \tag{11}$$

$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \hat{\mathbf{c}}_t \odot \mathbf{i}_t \tag{12}$$

$$\mathbf{h}_t = \tau(\mathbf{c}_t) \odot \mathbf{o}_t \tag{13}$$

where $\sigma(.)$ and $\tau(.)$ are the point-wise sigmoid and tanh functions. $\odot$ indicates element-wise multiplication. The first three are gating operations and the 4th one prepares the candidate information. The 5th operation updates the cell-state and finally in the 6th operation the output gate decided what to go into the current hidden state.

3. Unitary Evolution RNN (Arjovsky et al., 2016)

Unitary evolution RNN (uRNN) proposed to solve the vanishing and exploding gradients through a unitary recurrent matrix, which is for the form:

$$\mathbf{W} = \mathbf{D}_3\mathbf{R}_2\mathscr{F}^{-1}\mathbf{D}_2\Pi\mathbf{R}_1\mathscr{F}\mathbf{D}_1, \tag{14}$$

where

- $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3$: Diagonal matrices whose diagonal entries are of the from $\mathbf{D}_{kk} = e^{i\theta_k}$, implies each matrix have N parameters, $(\theta_0, \ldots, \theta_{N-1})$.
- $\mathscr{F}$ and $\mathscr{F}^{-1}$: Fast Fourier operator and inverse fast Fourier operator respectively.
- $\mathbf{R}_1, \mathbf{R}_2$: Householder reflections. $R = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^H}{\|\mathbf{v}\|}$, where $v \in \mathbb{C}^N$.

The total number of parameters for this uRNN operator is $7N$ and the matrix vector can be done $Nlog(N)$ time. It is parameter efficient and fast but not flexible and suffers from the retention of noise and difficulty in optimization due its unitarity.

4. Full Capacity Unitary RNN (Wisdom et al., 2016)

Full capacity unitary RNN (FC uRNN) does optimization on the full unitary set instead on a subset like uRNN. That is FC uRNN's recurrent matrix $\mathbf{W} \in U(N)$. There are several challenges in optimization over unitary manifold especially when combined with stochastic gradient method. The primary challenge being the optimization cost is $\mathcal{O}(N^3)$ per step.

5. Orthogonal RNN (Mhammedi et al., 2016)

Orthogonal RNN (oRNN) parametrizes the recurrent matrices using Householder reflections.

$$\mathbf{W} = \mathcal{H}_N(\mathbf{v}_N)...\mathcal{H}_{N-K+1}(\mathbf{v}_{N-k+1}), \tag{15}$$

where

$$\mathcal{H}_K(\mathbf{v}_K) = \begin{bmatrix} \mathbf{I}_{N-K} & 0 \\ 0 & \mathbf{I}_K - 2\frac{\mathbf{v}_K\mathbf{v}_K^H}{\|\mathbf{v}_K\|} \end{bmatrix} \tag{16}$$

and

$$\mathcal{H}_1(\mathbf{v}) = \begin{bmatrix} \mathbf{I}_{N-1} & 0 \\ 0 & \mathbf{v} \in \{-1, 1\} \end{bmatrix}. \tag{17}$$

where $\mathbf{v}_K \in \mathbb{R}^K$. The number of parameters in this parametrization is $\mathcal{O}(NK)$. When $N = K = 1$ and $v = 1$, it spans the rotation subset and when $v = -1$, it spans the full reflection subset.

6. Properties of Kronecker Product (Van Loan, 2000)

Consider a matrix $\mathbf{W} \in \mathbb{C}^{N\times N}$ factorized as a Kronecker product of F matrices $\mathbf{W}_0, \ldots, \mathbf{W}_{F-1}$,

$$\mathbf{W} = \mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{F-1} = \otimes_{i=0}^{F-1}\mathbf{W}_i. \tag{18}$$

Where each $\mathbf{W}_i \in \mathbb{C}^{P_i\times Q_i}$ respectively and $\prod_{i=0}^{f-1} P_i = \prod_{i=0}^{F-1} Q_i = N$. $\mathbf{W}_i$'s are called as Kronecker factors.

If the factors $\mathbf{W}_i$'s are $\left\{\begin{array}{l}\text{Nonsingular}\\ \text{Symmetric}\\ \text{Stochatsic}\\ \text{Orthogonal}\\ \text{Unitary}\\ \text{PSD}\\ \text{Toeplitz}\end{array}\right\}$ then $\mathbf{W}$ is $\left\{\begin{array}{l}\text{Nonsingular}\\ \text{Symmetric}\\ \text{Stochatsic}\\ \text{Orthogonal}\\ \text{Unitary}\\ \text{PSD}\\ \text{Block Toeplitz}\end{array}\right\}$

Theorem 1. *If* $\forall f \in 0, \ldots, F-1$, $\mathbf{W}_f$ *is unitary then* $\mathbf{W}$ *is also unitary.*

Proof.

$$\mathbf{W}^H\mathbf{W} = (\mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{f-1})^H(\mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{f-1}) \tag{19}$$

$$= (\mathbf{W}_0^H \otimes \cdots \otimes \mathbf{W}_{f-1}^H)(\mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{f-1}) \tag{20}$$

$$= \mathbf{W}_0^H\mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{f-1}^H\mathbf{W}_{f-1} = \mathbf{I}. \tag{21}$$

□

7. Bounds on the Spectrum of Kronecker Product using Soft Unitary Constraint

Consider a Kronecker factored square matrix $\mathbf{W} \in \mathbb{C}^{N\times N}$. That is $\mathbf{W} = \otimes_{f=0}^{F-1}\mathbf{W}_f$, where each $\mathbf{W}_f \in \mathbb{C}^{P_f\times P_f}$ respectively and $\prod_{f=0}^{F-1} P_f = N$. The theorem 1 suggests that if each of the factors is unitary then the Kronecker product $\mathbf{W} = \otimes_{f=0}^{F-1}\mathbf{W}_f$ is unitary. Soft unitary constraint enforces the unitary constraint approximately on the factors which results in an approximately unitary Kronecker product. Soft unitary constraint puts a Gaussian prior on the singular values of the Kronecker factor matrices. The expectation of this Gaussian is 1 and the variance is inversely proportional to the amplitude of the soft unitary constraint. The theorem below 2 shows the expectation and the variance on the spectrum of the Kronecker product as function of its factors.

Theorem 2. *The expected spectral norm and the condition number of a random Kronecker product whose random factors are approximately unitary is 1 and its variance is bounded by* $(1+\sigma^2)^F - 1$*, where* σ^2 *is the variance of the singular values of the Kronecker factors.*

Proof. If $\forall f \in 0, \ldots, F-1$, $\mathbf{W}_f$ is approximately unitary by using the soft unitary constraints, then the expected spectral norm: $\mathbb{E}[sn(\mathbf{W}_f)] = 1$ and the variance $\mathbb{V}[sn(\mathbf{W}_f)] = \sigma^2$, where σ^2 is inversely proportional to the amplitude of soft unitary constraints. $sn(\mathbf{W}_f)$ is a function which returns the spectral norm of the matrix $\mathbf{W}_f$.

$$\mathbb{E}[sn(\mathbf{W})] = \mathbb{E}[sn(\otimes_{f=0}^{F-1}\mathbf{W}_f)], \tag{22}$$

$$= \mathbb{E}[\prod_{f=0}^{F-1} sn(\mathbf{W}_f)], \tag{23}$$

$$= \prod_{f=0}^{F-1} \mathbb{E}[sn(\mathbf{W}_f)] = 1. \tag{24}$$

The equation 24 comes from the independence assumptions on the singular values across the factors. Similarly for the variance:

$$\mathbb{V}[sn(\mathbf{W})] = \mathbb{V}[sn(\otimes_{f=0}^{F-1}\mathbf{W}_f)], \tag{25}$$

$$= \mathbb{E}[\prod_{f=0}^{F-1} sn(\mathbf{W}_f)^2] - \mathbb{E}[\prod_{f=0}^{F-1} sn(\mathbf{W}_f)]^2, \tag{26}$$

$$= \prod_{f=0}^{F-1} \mathbb{E}[sn(\mathbf{W}_f)^2] - 1, \tag{27}$$

$$= (1+\sigma^2)^F - 1. \tag{28}$$

A similar result can be shown for the condition number. □

These are worst case results. In practice the results that we obtained on the spectral norm and the condition number of the Kronecker product is far better than the results predicted by the theory.

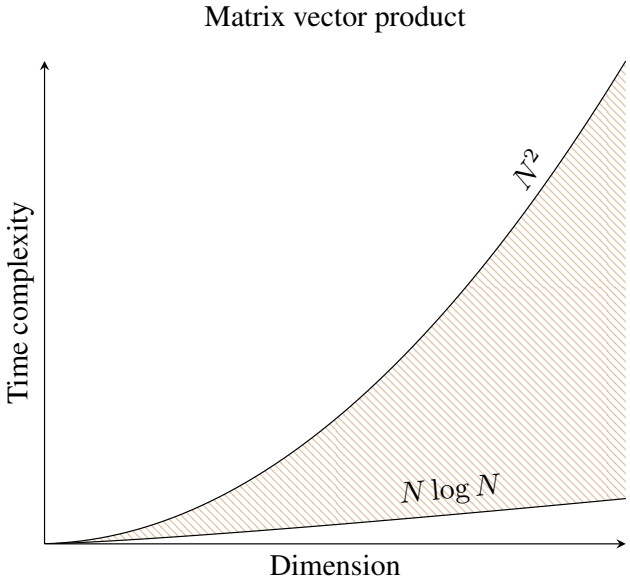

Figure 2. Graph illustrating the time complexity of dense vector product with Kronecker factored square matrices as a function of the vector dimension. Kronecker factorization allows a fine-grained control over the number of parameters and hence the computational efficiency. By choosing the number of factors and the size of each factors we can span all the Kronecker matrices in the shaded region of the graph and thus can control the amount of computation we want to invest in.

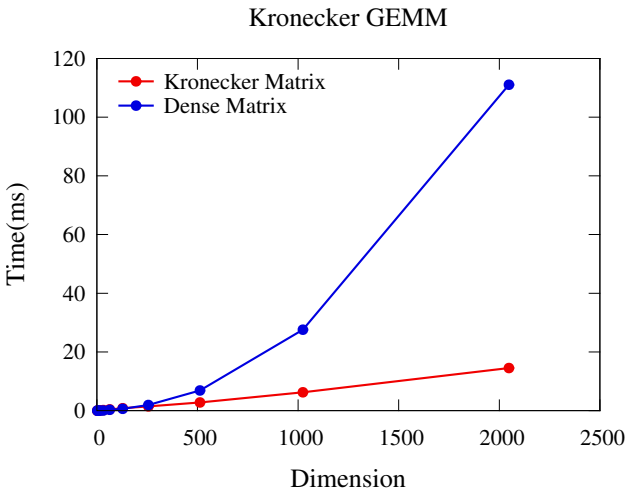

Figure 3. Comparison of matrix product between a dense matrix and a Kronecker matrix with 2x2 factors vs Dense matrix matrix product (GEMM). We use the standard BLAS notations (M = 64, Dimension = N = K).

8. Product between a Dense Matrix and a Kronecker Matrix

For simplicity here we use real number notations. Consider a dense matrix $\mathbf{X} \in \mathbb{R}^{M \times K}$ and a Kronecker factored matrix $\mathbf{W} \in \mathbb{R}^{N \times K}$. That is $\mathbf{W} = \otimes_{f=0}^{F-1} \mathbf{W}_f$, where each $\mathbf{W}_f \in \mathbb{R}^{P_f \times Q_f}$ respectively and $\prod_{f=0}^{F-1} P_f = N$ and $\prod_{f=0}^{F-1} Q_f = K$. Let us illustrate the matrix product $\mathbf{X}\mathbf{W}^T$ resulting in a matrix $\mathbf{Y} \in \mathbb{R}^{M \times N}$.

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^T. \tag{29}$$

The computational complexity first expanding the Kronecker factored matrix and then computing the matrix product is $\mathcal{O}(MNK)$. This can be reduced by exploiting the recursive definition of Kronecker matrices. For examples when $N = K$ and $\forall_f \{P_f = Q_f = 2\}$, the matrix product can be computed in $\mathcal{O}(MN \log N)$ time instead of $\mathcal{O}(MN^2)$.

The matrix product in 29 can be recursively defined as

$$\mathbf{Y} = (\dots(\mathbf{X} \odot \mathbf{W}_0^T) \otimes \cdots \otimes \mathbf{W}_{F-1}^T). \tag{30}$$

Please note that the binary operator $\odot$ is not the standard matrix multiplication operator but instead it denotes a strided matrix multiplication. The stride is computed according to the algebra of Kronecker matrices. Let us define $\mathbf{Y}$ recursively:

$$\mathbf{Y}_0 = \mathbf{X} \odot \mathbf{W}_0 \tag{31}$$
$$\mathbf{Y}_f = \mathbf{Y}_{f-1} \odot \mathbf{W}_f. \tag{32}$$

Combining equation 34 and 32

$$\mathbf{Y} = \mathbf{Y}_{F-1} = (\dots(\mathbf{X} \odot \mathbf{W}_0^T) \otimes \cdots \otimes \mathbf{W}_{F-1}^T). \tag{33}$$

We use the above notation for $\mathbf{Y}$ in the algorithm. That is the algorithm illustrated here will cache all the intermediate outputs $(\mathbf{Y}_0, \dots, \mathbf{Y}_{F-1})$ instead of just $\mathbf{Y}_{F-1}$. These intermediate outputs are then later to compute the gradients during the back-propagation. This cache will save some computation during the back-propagation. If the model is just being used for inference then the algorithm can the organized in such a way that we do not need to cache the intermediate outputs and thus save memory.

Algorithm for computing the product between a dense matrix and a Kronecker factored matrix34 is given below 1. All the matrices are assumed to be stored in row major order. For simplicity the algorithm is illustrated in a serial fashion. Please note the lines 4 to 15 except lines 9-11 can be trivially parallelized as it writes to independent memory locations. The GPU implementation exploits this fact.

Algorithm 1. Dense matrix product with a Kronecker matrix, $\mathbf{Y} = (\dots(\mathbf{X}\mathbf{W}_0^T) \otimes \cdots \otimes \mathbf{W}_{F-1}^T)$

Input: Dense matrix $\mathbf{X} \in \mathbb{R}^{M \times K}$, Kronecker factors $\{\mathbf{W}_0, \dots, \mathbf{W}_{F-1}\} : \mathbf{W}_f \in \mathbb{R}^{p_f \times q_f}$, Size of each Kronecker factors $\{(P_0, Q_0), \dots, (P_{F-1}, Q_{F-1})\} : \prod_{f=0}^{F-1} P_f = N, \prod_{f=0}^{F-1} Q_f = K$,
Output: Output matrix $Y_{F-1} \in \mathbb{R}^{M \times N}$

1: **for** $f = 0$ to $F - 1$ **do**
2: $stride = K/Q_f$
3: $index = 0$
4: **for** $m = 0$ to $M - 1$ **do**
5: $\mathbf{X}_m = \mathbf{X} + m \times K$
6: **for** $p = 0$ to $P_f - 1$ **do**
7: **for** $s = 0$ to $stride - 1$ **do**
8: $\mathbf{Y}_f[index] = 0$
9: **for** $q = 0$ to $Q_k - 1$ **do**
10: $\mathbf{Y}_f[index] = \mathbf{Y}_f[index] + \mathbf{X}_m[q \times stride + s] \times \mathbf{W}_f[p \times Q_f + q]$
11: **end for**
12: $index = index + 1$
13: **end for**
14: **end for**
15: **end for**
16: $K = stride$
17: $M = M \times P_f$
18: $X = Y_f$
19: **end for**

9. Gradient Computation in a Kronecker Layer

Following the notations from the above section 8, here we illustrate the algorithm for computing the gradients in a Kronecker layer. To be clear and concrete the Kronecker layer does the following computation in the forward pass 32.

$$\mathbf{Y} = \mathbf{Y}_{F-1} = (\dots(\mathbf{X} \odot \mathbf{W}_0^T) \otimes \cdots \otimes \mathbf{W}_{F-1}^T). \tag{34}$$

That is, the Kronecker layer is parametrized by a Kronecker factored matrix $\mathbf{W} = \otimes_{f=0}^{F-1} \mathbf{W}_f$ stored as it factors $\{\mathbf{W}_0, \dots, \mathbf{W}_{F-1}\}$ and it takes an input $\mathbf{X}$ and produces output $\mathbf{Y} = \mathbf{Y}_{F-1}$ using the algorithm 1.

The following algorithm 2 computes the Gradient of the Kronecker factors: $\{\mathbf{gW}_0, \dots, \mathbf{gW}_{F-1}\}$ and the Jacobian of the input matrix $\mathbf{gX}$ given the Jacobian of the output matrix: $\mathbf{gY} = \mathbf{gY}_{F-1}$.

Algorithm 2. Gradient computation in a Kronecker layer.

Input: Input matrix $\mathbf{X} \in \mathbb{R}^{M \times K}$, Kronecker factors $\{\mathbf{W}_0, \dots, \mathbf{W}_{F-1}\} : \mathbf{W}_f \in \mathbb{R}^{p_f \times q_f}$, Size of each Kronecker factors $\{(P_0, Q_0), \dots, (P_{F-1}, Q_{F-1})\} : \prod_{f=0}^{F-1} P_f = N, \prod_{f=0}^{F-1} Q_f = K$, All intermediate output matrices from the forward pass: $\{\mathbf{Y}_0, \dots, \mathbf{Y}_{F-1}\}$, Jacobian of output matrix: $\mathbf{gY}_{F-1} \in \mathbb{R}^{M \times N}$

Output: Gradient of Kronecker factors: $\{\mathbf{gW}_0, \dots, \mathbf{gW}_{F-1}\}$ and Jacobian of input matrix: $\mathbf{gX} \in \mathbb{R}^{M \times N}$.

$T = M \times N$
$strideP = 1$
$strideQ = 1$
$\mathbf{gY} = \mathbf{gY}_{F-1}$
for $f = F - 1$ to 0 **do**
  $R = strideP \times P_f$
  $S = strideQ \times Q_f$
  $T = T / P_f$
  $\mathbf{Z} = nullptr$
  $\mathbf{gZ} = nullptr$
  if $f == 0$ **then**
    $\mathbf{Z} = \mathbf{X}$
    $\mathbf{gZ} = \mathbf{gX}$
  else
    $\mathbf{gZ} = \mathbf{Y}_{f-1}$
    $\mathbf{Z} = \mathbf{gZ}$
  end if
  $index = 0$
  for $t = 0$ to $T - 1$ **do**
    $\mathbf{Z}_t = \mathbf{Z} + t \times S$
    for $p = 0$ to $P_f - 1$ **do**
      for $s = 0$ to $strideQ - 1$ **do**
        for $q = 0$ to $Q_k - 1$ **do**
          $\mathbf{gW}_f[p \times Q_k + 1] = \mathbf{gW}_f[p \times Q_k + 1] + \mathbf{Z}_t[q \times strideQ + s] \times \mathbf{gY}[index]$
        end for
        $index = index + 1$
      end for
    end for
  end for
  $index = 0$
  for $t = 0$ to $T - 1$ **do**
    $\mathbf{gY}_t = \mathbf{gY} + t \times R$
    for $p = 0$ to $P_f - 1$ **do**
      for $s = 0$ to $strideQ - 1$ **do**
        $\mathbf{gZ}[index] = 0$
        for $q = 0$ to $Q_k - 1$ **do**
          $\mathbf{gZ}[index] = \mathbf{gZ}[index] + \mathbf{gY}[q \times strideQ + s] \times \mathbf{W}_f[q \times P_f + q]$
        end for
        $index = index + 1$
      end for
    end for
  end for
  $\mathbf{gY} = \mathbf{gZ}$ //We reuse the memory for the intermediate outputs to store the gradients.
  $strideQ = S$
  $strideP = R \times Q_f / P_f$
end for

References

Arjovsky, Martin, Shah, Amar, and Bengio, Yoshua. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Mhammedi, Zakaria, Hellicar, Andrew, Rahman, Ashfaqur, and Bailey, James. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. *arXiv preprint arXiv:1612.00188*, 2016.

Van Loan, Charles F. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1):85–100, 2000.

Wisdom, Scott, Powers, Thomas, Hershey, John, Le Roux, Jonathan, and Atlas, Les. Full-capacity unitary recurrent neural networks. In *Advances In Neural Information Processing Systems*, pp. 4880–4888, 2016.