Optimal Kronecker-Sum Approximation of Real Time Recurrent Learning

Frederik Benzing [*1] **Marcelo Matheus Gauy** [*1] **Asier Mujika** [1] **Anders Martinsson** [1] **Angelika Steger** [1]

Abstract

One of the central goals of Recurrent Neural Networks (RNNs) is to learn long-term dependencies in sequential data. Nevertheless, the most popular training method, Truncated Backpropagation through Time (TBPTT), categorically forbids learning dependencies beyond the truncation horizon. In contrast, the online training algorithm Real Time Recurrent Learning (RTRL) provides untruncated gradients, with the disadvantage of impractically large computational costs. Recently published approaches reduce these costs by providing noisy approximations of RTRL. We present a new approximation algorithm of RTRL, Optimal Kronecker-Sum Approximation (OK). We prove that OK is optimal for a class of approximations of RTRL, which includes all approaches published so far. Additionally, we show that OK has empirically negligible noise: Unlike previous algorithms it matches TBPTT in a real world task (character-level Penn TreeBank) and can exploit online parameter updates to outperform TBPTT in a synthetic string memorization task.
Code available at GitHub.

1. Introduction

Learning to predict sequential and temporal data is one of the core problems of Machine Learning arising for example in language modeling, speech generation and Reinforcement Learning. One of the main aims when modeling sequential data is to capture long-term dependencies. Most of the significant advances towards this goal have been achieved through Recurrent Neural Nets (RNNs). More specifically, different architectures (e.g. the Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and the Recurrent Highway Network (RHN) (Zilly et al., 2017)) were developed to facilitate learning long-term dependencies and achieved notable successes. However, few improvements have been made regarding the training methods of RNNs. Since Williams and Peng (1990) developed Truncated Backpropagation through Time (TBPTT), it continues to be the most popular training method in many areas (Mnih et al., 2016; Mehri et al., 2017; Merity et al., 2018) - despite the fact that it does not seem to align well with the goal of learning arbitrary long-term dependencies. This is because TBPTT 'unrolls' the RNN only for a fixed number of time steps T (the truncation horizon) and backpropagates the gradient for these steps only. This almost categorically forbids learning dependencies beyond the truncation horizon. Unfortunately, extending the truncation horizon makes TBPTT increasingly memory consuming, since long input sequences need to be stored, and considerably slows down learning, since parameters are updated less frequently, a phenomenon known as 'update lock' (Jaderberg et al., 2017).

An alternative avoiding these issues of TBPTT is Real Time Recurrent Learning (RTRL) (Williams & Zipser, 1989). The advantages of RTRL are that it provides untruncated gradients, which in principle allow the network to learn arbitrarily long-term dependencies, and that it is fully online, so that parameters are updated frequently allowing faster learning. However, its runtime and memory requirements scale poorly with the network size and make RTRL infeasible for practical applications. As a remedy to this problem, Tallec and Ollivier (2017a) proposed replacing the full gradient of RTRL by an unbiased, less computationally costly but noisy approximation (Unbiased Online Recurrent Optimisation, UORO). Recently, Mujika et al. (2018) reduced the noise of this approach and demonstrated empirically that their improvement (Kronecker factored RTRL, KF-RTRL) allows learning complex real world data sets (character-level Penn TreeBank (Marcus et al., 1993)). Nevertheless, the noise introduced by KF-RTRL remains a problem, leading to slower learning and worse performance when compared to TBPTT.

To address this problem, we propose a new approximation of RTRL, Optimal Kronecker-Sum Approximation (OK). Extending ideas of KF-RTRL, it approximates the gradient by a sum of Kronecker-factors. It then introduces a novel procedure to perform the online updates of this approximation. We prove that this procedure has minimum achievable variance for a certain class of approximations, which includes UORO and KF-RTRL. Thus, OK does not

[*]Equal contribution [1]Department of Computer Science, ETH Zurich, Zurich, Switzerland. Correspondence to: FB <benzingf@inf.ethz.ch>, MMG <marcelo.matheus@inf.ethz.ch>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

only improve all previous approaches but explores the theoretical limits of the current approximation class. Empirically, we demonstrate that OK reduces the noise to a negligible level: In contrast to previous algorithms, OK matches the performance of TBPTT on a standard RNN benchmark (character-level Penn TreeBank) and also outperforms it on a synthetic string memorization task exploiting online parameter updates. Similarly to KF-RTRL, OK is applicable to a subclass of RNNs. Besides standard RNNs this includes LSTMs and RHNs thereby covering some of the most widely used architectures.

Our theoretical findings include a construction (and proof) of a minimum-variance unbiased low-rank approximator of an arbitrary matrix, which might be applicable in other contexts of Machine Learning relying on unbiased gradients.

As a more exploratory contribution, we develop another algorithm, Kronecker-Triple-Product (KTP). Its main novelty is to match the runtime and memory requirements of TBPTT, even when measured per batch-element. Our experiments show that KTP is more noisy than OK, but that it can learn moderate time dependencies. In addition, we design an experiment to suggest directions for further improvements.

2. Related Work

The most prominent training method for RNNs is Truncated Backpropagation through Time (TBPTT) (Williams & Peng, 1990), often yielding good results in practice. It calculates truncated gradients forbidding the network to learn long-term dependencies beyond the truncation horizon. The untruncated version of this algorithm, Backpropagation Trough Time (BPTT) (Rumelhart et al., 1986), stores all past inputs and unrolls the network from the first time step, often making its computational cost unmanageable.

We now review some alternatives to TBPTT. Besides Real Time Recurrent Learning and its approximations, which will be described in detail in the next section, this includes Anticipated Reweighted Backpropagation (Tallec & Ollivier, 2017b) which samples different truncation horizons and weights the obtained gradients to calculate an overall unbiased gradient. Sparse Attentative Backtracking (Ke et al., 2018) uses an attention mechanism (Vaswani et al., 2017) and propagates the gradient along paths with high attention to extend the time span of learnable dependencies.

Other ideas avoid unrolling the network. For example, Decoupled Neural Interfaces (Jaderberg et al., 2017) use neural nets to learn to predict future gradients, while Ororbia et al. (2018) propose a predictive coding based approach.

For RNNs where the hidden state converges, it is also possible to avoid BPTT as shown for example by Recurrent Backpropagation (Liao et al., 2018) and the closely related Equilibrium Propagation (Scellier & Bengio, 2017).

Another approach fixes the recurrent weights and only trains the output weights. This is known as Reservoir computing (Lukoševičius & Jaeger, 2009) and was applied for example in (Jaeger, 2001; Maass et al., 2002).

3. RTRL and its Approximations

In this section, we derive Real Time Recurrent Learning (RTRL) (Williams & Zipser, 1989) and provide a common framework describing approximation algorithms for RTRL. We then embed previous algorithms and our contribution into this framework. The class of approximators for which OK is optimal, as well as a precise optimality statement are given in section 3.4, Definition 1 and Theorem 1. The section concludes with a theoretical comparison of the different approximation algorithms including concrete examples illustrating the advantages of OK.

Since the two main goals of approximating RTRL are (a) providing unbiased estimates of the gradient with (b) as little noise as possible, we make these notions precise. For a matrix A and a random variable A', we say that A' is an unbiased approximator of A if $\mathrm{E}[A'] = A$ and define the noise/variance of A' to be $\mathrm{Var}[A'] = \mathrm{E}\left[\|A' - A\|^2\right]$, where we use the Frobenius norm for matrices.

3.1. RTRL and a General Approximation Framework

We start by formally defining RNNs before deriving RTRL. A RNN maintains a hidden state h_t across several time steps. The next hidden state h_{t+1} is computed as a differentiable function f of h_t, the input x_{t+1} and a set of learnable parameters θ, $h_{t+1} = f(x_{t+1}, h_t, \theta)$. Predictions for the desired output (for example predicting the next character of a given text) are a function of h_t and θ. We aim to minimize some loss function L_t of our predictions and therefore compute $\frac{dL_t}{d\theta}$ to perform gradient descent on θ.

To derive RTRL, we use the chain rule to rewrite $\frac{dL_t}{d\theta} = \frac{dL_t}{dh_t}\frac{dh_t}{d\theta}$. Next we observe

$$\frac{dh_t}{d\theta} = \frac{\partial h_t}{\partial x_t}\frac{dx_t}{d\theta} + \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{d\theta} + \frac{\partial h_t}{\partial \theta}\frac{d\theta}{d\theta}.$$

Writing $G_t := \frac{dh_t}{d\theta}$, $H_t = \frac{\partial h_t}{\partial h_{t-1}}$ and $F_t = \frac{\partial h_t}{\partial \theta}$, this simplifies to (note $dx_t/d\theta = 0$ as x_t does not depend on θ)

$$G_t = H_t G_{t-1} + F_t. \tag{1}$$

RTRL simply calculates and stores G_t at each time step using the recurrence (1) and uses it to calculate $\frac{dL}{d\theta} = \frac{dL_t}{dh_t}\frac{dh_t}{d\theta}$. This shows that RTRL is fully online and can perform frequent parameter updates.

However, we already see why RTRL is impractical for applications. For a standard RNN with n hidden units and

n^2 parameters, G_t has dimensions $n \times n^2$ and we need to evaluate the matrix multiplication $H_t G_{t-1}$, so that RTRL requires memory n^3 and runtime n^4 per batch element. This contrasts with TBPTT, which needs memory Tn and runtime Tn^2, where T is the truncation horizon (Williams & Peng, 1990). To make RTRL competitive, we therefore need to find computationally efficient approximations.

The core of RTRL is the recurrence equation (1) and it should be the focus of any approximation. Previous approaches to approximate RTRL can be summarised as follows (see also Algorithm 1): Firstly, decide on a format in which the approximator G'_t of the gradient G_t is stored. This format should require less memory than storing all $n \times n^2$ numbers explicitly. Secondly, assuming G'_{t-1} is given in the desired format, bring $H_t G'_{t-1}$ and F_t in this format. Thirdly, 'mix' the two terms $H_t G'_{t-1}$ and F_t, to bring their sum in the desired format.

Algorithm 1 One step of unbiasedly approximating RTRL.
This algorithm describes a framework for approximating RTRL. It assumes that the approximation is stored in a given format, called $\circledast$, and that routines $Ap(\cdot), Mix(\cdot,\cdot)$ for bringing (sums of) matrices into this format are known.

Input: input x_t, hidden state h_{t-1}, parameters θ, unbiased approximator G'_{t-1} of G_{t-1} stored in prescribed format $\circledast$.
Output: hidden state h_t, unbiased approximator G'_t of G_t in format $\circledast$.
/* Preliminary calculations */
$h_t \leftarrow$ new hidden state, based on h_{t-1}, θ, x_t
$H \leftarrow \frac{\partial h_t}{\partial h_{t-1}}$, $F \leftarrow \frac{\partial h_t}{\partial \theta}$
/* Bring addends HG'_{t-1}, F_t in desired format $\circledast$*/
$A_1 \leftarrow Ap(HG'_{t-1})$, where $Ap(x)$ is an unbiased approximator of x in format $\circledast$
$A_2 \leftarrow Ap(F)$
/* Mix two addends A_1, A_2 */
$G'_t \leftarrow Mix(A_1, A_2)$, where $Mix(x,y)$ is an unbiased approximator of $x+y$ in format $\circledast$

In order to obtain convergence guarantees for gradient descent with noisy gradients, it is crucial that the approximator G'_t be unbiased and we therefore make all approximations unbiased. In the appendix (A.2.3), we empirically evaluate the difference between biased and unbiased approximators.

Another important consideration for the convergence of RTRL and its approximation is the stability (boundedness) of gradients and noise, which could in principle accumulate indefinitely over time. Under reasonable assumptions, which are standard in order to avoid the exploding gradient issue in RNNs (Pascanu et al., 2013), it is shown in Theorem 1 of (Mujika et al., 2018) that the approximations of RTRL are stable over time. This result applies to all the approximations presented below.

3.2. Unbiased Online Recurrent Optimization (UORO)

We now present UORO (Tallec & Ollivier, 2017a), the first approximation algorithm of RTRL. UORO follows the framework described above. It stores the approximation G'_{t-1} as the outer-product (or Kronecker-product) of two vectors[1] u_{t-1}, v_{t-1} of dimensions n and n^2, i.e. $G'_{t-1} = u_{t-1} \otimes v_{t-1}$. Next, it rewrites $H_t G'_{t-1}$ observing $H_t(u_{t-1} \otimes v_{t-1}) = (H_t u_{t-1}) \otimes v_{t-1} = \overline{u}_{t-1} \otimes v_{t-1}$. We omit the details of approximating F_t by a product $r_t \otimes s_t$ and simply note that this process creates noise. We now explain how the two terms $\overline{u}_{t-1} \otimes v_{t-1}$ and $r_t \otimes s_t$ are 'mixed' in an unbiased way. This can be achieved by the so called 'sign-trick'. This means choosing a uniformly random sign $c \in \{\pm 1\}$ and writing[2] $G'_t = (\overline{u}_{t-1} + c \cdot r_t) \otimes (v_{t-1} + c \cdot s_t)$. A simple calculation shows

$$E\big[(\overline{u}_{t-1} + c \cdot r_t) \otimes (v_{t-1} + c \cdot s_t)\big] = \overline{u}_{t-1} \otimes v_{t-1} + r_t \otimes s_t,$$

so that G'_{t+1} is an unbiased approximator of $H_t G'_{t-1} + F_t$. Induction on t and linearity of expectation now show that G'_t is an unbiased approximator of G_t. It is easily checked that UORO needs runtime and memory of order n^2.

3.3. Kronecker Factored RTRL (KF-RTRL)

The algorithm KF-RTRL (Mujika et al., 2018) is similar in spirit to UORO. The main difference is that it approximates G'_t as the Kronecker-product of a vector $u_t \in \mathbb{R}^{1 \times n}$ and a matrix $A_t \in \mathbb{R}^{n \times n}$, i.e. $G'_t = u_t \otimes A_t$. While this looks equivalent to UORO at first glance, Mujika et al. observed that for many RNN architectures, including standard RNNs, RHNs and LSTMs, it is possible to factor F_t as a Kronecker-product $F_t = h_t \otimes D_t$, without adding any noise. Here, D_t is a diagonal matrix. Similarly to UORO, we can exploit properties of the Kronecker-product to rewrite $H_t G'_{t-1} = u_{t-1} \otimes (H_t A_{t-1})$ in the desired format and use a sign trick to mix the two addends $H_t G'_{t-1}$ and F_t to obtain G'_t.

Note that KF-RTRL has memory requirements of roughly n^2 for storing the matrix A_t and runtime of order n^3 due to the matrix-matrix multiplication $H_t A_{t-1}$. No additional memory is required to obtain $\frac{dL_t}{d\theta}$ as we can write $\frac{dL_t}{d\theta} = \frac{dL_t}{dh_t} \cdot G'_t = \frac{dL_t}{dh_t} \cdot (u_t \otimes A_t) = u_t \otimes \left(\frac{dL_t}{dh_t} A_t\right)$.

3.4. Optimal Kronecker-Sum Approximation (OK)

We now describe our algorithm OK. The calculations carried out by OK are reasonably simple as can be seen in

[1] For concreteness, we consider a standard RNN with n hidden units and n^2 parameters, so that G_t has dimension $n \times n^2$.

[2] We remark that UORO additionally introduces a variance reduction technique which rescales the factors of each outer-product to have the same norm.

the pseudo-code below. However, their correctness is not immediate and relies on the proof given in the appendix. We give some intuition and concrete examples illustrating the improvements of OK in Section 3.6, which can be read independently of the detailed implementation.

Let us start by briefly reconsidering the previous two algorithms. In our framework, there are two noise sources for approximating RTRL. Firstly, we rewrite addends F_t and H_tG_{t-1} in the desired format and secondly, we mix them. The first noise source is eliminated by KF-RTRL since it factors $F_t = h_t \otimes D_t$ and $H_tG'_{t-1} = u_{t-1} \otimes (H_tA_{t-1})$ noiselessly. The second noise source, stemming from mixing the terms, is the focus of our algorithm, and we shall prove below that OK not only improves previous algorithms in this step but has minimum achievable variance.

We also note that UORO and KF-RTRL both approximate G_t by a '1-Kronecker-Sum' G'_t as defined below.

Definition 1 (Kronecker-Sum, format)**.** *For a matrix $G \in \mathbb{R}^{m\times n}$, we say that G is given as a r-Kronecker-Sum, if we are given $u_1, \dots, u_r \in \mathbb{R}^{a\times b}$ and $A_1, \dots, A_r \in \mathbb{R}^{c\times d}$ with $ac = m, bd = n$ so that $G = \sum_{i=1}^r u_i \otimes A_i$. We refer to (a, b, c, d) as the format of the Kronecker-Sum.*

3.4.1. Outline of OK

Our new algorithm, OK, has a parameter r, and approximates G_t by a r-Kronecker-Sum, where each summand is the product of a vector $u_i \in \mathbb{R}^{1\times n}$ and a matrix $A_i \in \mathbb{R}^{n\times n}$, similar to KF-RTRL. Concretely, we have $G'_{t-1} = \sum_{i=1}^r u_i \otimes A_i$. We will refer to the algorithm as r-OK, or simply OK depending on the context. Usually, r is a small constant and much smaller than the network size n.

Analogously to KF-RTRL, we focus on situations where $F_t = h_t \otimes D_t$ can be factored as a Kronecker product. A precise condition (Mujika et al., 2018)[Lemma 1] for when this is possible is given in the appendix (A.0.1). When we can factor $F_t = h_t \otimes D_t$, the remaining task for OK is to unbiasedly approximate the $(r + 1)$-Kronecker-Sum

$$G = u_1 \otimes (H_tA_1) + \ldots + u_r \otimes (H_tA_r) + h \otimes D \qquad (2)$$

by a r-Kronecker-Sum G'_t. Equivalently, OK finds random vectors $u'_1, \dots, u'_r$ and matrices $A'_1, \dots, A'_r$ so that for

$$G' = \sum_{i=1}^{r} u'_i \otimes A'_i$$

we have $\mathrm{E}[G'] = G$. We now state the main optimality property of our algorithm.

Theorem 1. *Let G be an $(r+1)$-Kronecker-Sum and let G' be the random r-Kronecker-Sum constructed by OK. Then G' unbiasedly approximates G. Moreover, for any random r-Kronecker-Sum Y of the same format as G' which satisfies $\mathrm{E}[Y] = G$, it holds that $\mathrm{Var}[Y] \geq \mathrm{Var}[G']$.*

We defer the proof to the appendix and only describe the main ideas for constructing G'. The first step, carried out in Algorithm 2, is to use linear algebra to reduce the problem to the following: Given a matrix $C \in \mathbb{R}^{(r+1)\times(r+1)}$, find a minimum-variance, unbiased approximator C' of C, so that the (matrix-)rank of C' is always at most r, i.e. C' can be factored as $L'R'^T$ for some $L', R' \in \mathbb{R}^{(r+1)\times r}$. The next two steps are handled by Algorithm 3: It calculates the singular value decomposition (SVD) of C, so that it remains to approximate a diagonal matrix D, and then constructs an optimal approximator D' of D. In the appendix, we give a duality argument to prove that D' is indeed optimal.

In total, the runtime of OK is of order rn^3, due to performing r matrix-matrix multiplications (see equation (2)), and the memory requirement is of order rn^2. The cost of calculating the optimal approximator G' is asymptotically negligible.

We also state the following, more general theorem. It might be useful in other settings where unbiased gradient approximations are important. Its proof is given in the appendix.

Theorem 2. *Given $C \in \mathbb{R}^{m\times n}$ and $r \leq \min\{m, n\}$, one can (explicitly) construct an unbiased approximator C' of C, so that C' always has rank at most r, and so that C' has minimal variance among all such unbiased, low-rank approximators. This can be achieved asymptotically in the same runtime as computing the SVD of C.*

3.4.2. Details of OK

Here, we present pseudo-code for OK (Algorithm 2). We make use of the algorithm SVD, a standard Linear Algebra algorithm (Golub & Van Loan, 1996; Cline & Dhillon, 2006) calculating the singular value decomposition of a matrix C. SVD finds a diagonal matrix D and orthogonal matrices U, V so that $C = UDV^T$. We only apply SVD to 'small' matrices $C \in \mathbb{R}^{(r+1)\times(r+1)}$, where it needs runtime $O\left(r^3\right)$ and memory $O\left(r^2\right)$.

3.5. Kronecker Triple Product (KTP)

Finally, we present another, more exploratory algorithm approximating RTRL still following the framework from Algorithm 1. KTP approximates G_t by a sum of r Kronecker-triple-products, i.e. $G'_t = \sum_{i=1}^r a_i \otimes b_i \otimes c_i$ where $a_i, c_i \in \mathbb{R}^{1\times n}$ and $b_i \in \mathbb{R}^{n\times 1}$. Before describing the remaining details of KTP, we motivate the suggested changes: On the one hand, KTP only requires memory of order rn rather than n^2 for each batch element. On the other hand, when computing H_tG_{t-1} we can write each of the addends as $H_t(a_i \otimes b_i \otimes c_i) = a_i \otimes (H_tb_i) \otimes c_i$. Computing H_tb_i only[3] takes time n^2, as opposed to time n^3 for the matrix-matrix multiplications of KF-RTRL and OK. Thus, KTP

[3]It is possible to evaluate H_tb without storing H_t for each batch element, see appendix A.0.3.

Algorithm 2 The OK approximation

Input: Vectors $u_1, \dots, u_{r+1}$ and matrices $A_1, \dots, A_{r+1}$
Output: Random vectors $u'_1, \dots, u'_r$ and matrices $A'_1 \dots A'_r$, such that $\sum_{i=1}^r u'_i \otimes A'_i$ is an unbiased, minimum-variance approximator of $\sum_{i=1}^{r+1} u_i \otimes A_i$
/*Rewrite in terms of orthonormal basis (onb)*/
$v_1, \dots, v_{r+1} \leftarrow$ onb of span$\{u_1, \dots, u_{r+1}\}$
$B_1, \dots, B_{r+1} \leftarrow$ onb span$\{A_1, \dots, A_{r+1}\}$
for $1 \le i, j \le r+1$ **do**
  $L_{i,j} \leftarrow \langle v_i, u_j \rangle, \quad R_{i,j} \leftarrow \langle B_i, A_j \rangle$
end for
/*Find optimal rank r approximation of matrix C */
$C \leftarrow LR^T$
$(L', R') \leftarrow Opt(C)$ {see Algorithm 3 for $Opt(\cdot)$}
/*Generate output*/
for $1 \le j \le r$ **do**
  $u'_j \leftarrow \sum_{i=1}^{r+1} L'_{i,j} v_i, \quad A'_j \leftarrow \sum_{i=1}^{r+1} R'_{i,j} B_i$
end for

Algorithm 3 $Opt(C)$

Input: Matrix $C \in \mathbb{R}^{(r+1)\times(r+1)}$
Output: Random matrices $L', R' \in \mathbb{R}^{(r+1)\times r}$, so that $L'R'^T$ is an unbiased, min-variance approximator of C
/* Reduce to diagonal matrix D*/
$(D, U, V) \leftarrow \mathrm{SVD}(C)$
$(d_1, \dots, d_{r+1}) \leftarrow$ diagonal entries of D
/* Find approximator ZZ^T for small d_i $(i \ge m)$*/
$m \leftarrow \min\{i \colon (r-i+1)d_i \le \sum_{j=i}^r d_j\}$
$s_1 \leftarrow \sum_{i=m}^{r+1} d_i, \quad k \leftarrow r-m+1$
$z_0 \leftarrow \left(\sqrt{1-\frac{d_m k}{s_1}}, \dots, \sqrt{1-\frac{d_{r+1} k}{s_1}}\right)^T \in \mathbb{R}^{(k+1)\times 1}$
$z_1, \dots, z_k \leftarrow$ so that $z_0, z_1, \dots, z_k$ is an onb of $\mathbb{R}^{(k+1)\times 1}$
$s \leftarrow$ vector of $k+1$ uniformly random signs
$Z \leftarrow \sqrt{\frac{s_1}{k}} \cdot (s \odot z_1, \dots, s \odot z_k)$ {*pointwise product* $\odot$}
/* Initialise L', R' to approximate D*/
$L', R' \leftarrow \mathrm{diag}(\sqrt{d_1}, \dots, \sqrt{d_{m-1}}, Z)$ {*Block-diagonal*}
/*Approximate $C = UDV^T$*/
$L' \leftarrow UL', \quad R' \leftarrow VR'$

matches the memory and runtime of TBPTT.

We now describe the remaining details of KTP. In order to bring $H_t G_{t-1}$ into the original format, we simply rewrite $H_t(a_i \otimes b_i \otimes c_i) = a_i \otimes (H_t b_i) \otimes c_i = a_i \otimes \bar{b}_i \otimes c_i$. To bring F_t in the same format, we again make use of the fact, that F_t can be factored as $F_t = h \otimes D$ where D is a diagonal matrix. This allows us to easily find an optimal, unbiased rank-r approximator $D' = \sum_{i=1}^r d_i \otimes d_i^T$ of D, where the d_i are random vectors constructed with Algorithm 3. Note that this algorithm is similar to the original UORO approach, but uses its knowledge about D in order to construct an optimal (rather than non-optimal) low-rank approximator of D and in order to reduce memory requirements from n^2 to n.
All in all, we have rewritten $F'_t = \sum_{i=1}^r h \otimes d_i \otimes d_i^T$ in the desired format. It remains to mix the two addends $H_t G'_{t-1}$ and F'_t. To this end, we mix, for each i, the i-th summands of $H_t G'_{t-1}$ and F'_t using a generalisation of the sign trick: We choose a vector of three signs $(s_1, s_2, s_1 s_2)$, where s_1, s_2 are uniform and independent, and approximate $a_i \otimes \bar{b}_i \otimes c_i + h \otimes d_i \otimes d_i^T$ by $(a_i + s_1 \cdot h) \otimes (\bar{b}_i + s_2 \cdot d_i) \otimes (c_i + s_1 s_2 \cdot d_i^T)$, so that altogether we obtain

$$G'_t = \sum_{i=1}^r (a_i + s_1 \cdot h) \otimes (\bar{b}_i + s_2 \cdot d_i) \otimes (c_i + s_1 s_2 d_i^T).$$

The 'mixing' procedure presented above is based on heuristics. We show in the appendix (A.0.2) that heuristics are somewhat necessary since the concept of an 'optimal' approximator is not well-defined in this case and related to NP-hard problems (Hillar & Lim, 2013).

3.6. Comparison

When comparing different unbiased approximations of RTRL, the focus lies on comparing noise/variance of the respective approximators, since this determines the speed of convergence and the final performance. To make comparisons as fair as possible we will also consider the different runtime and memory requirements, see also Table 1 for an overview. Here, we focus on theoretical considerations. Experimental evaluations of the noise will be presented in the next section.

In (Mujika et al., 2018) it was already shown that KF-RTRL has significantly less noise than UORO, so that we shall focus on KF-RTRL and OK only.

We first compare 1-OK to KF-RTRL, as the memory and runtime requirements for these two algorithms are asymptotically equal. The difference between the two algorithms is how they mix a sum of two Kronecker-products to obtain one Kronecker-product. From Theorem 1 it is immediate that OK performs at least as well as KF-RTRL. In general, it depends on the two Kronecker-products, which need to be mixed, how much better OK performs than KF-RTRL. We show two extreme cases here - the 'average' case arising during learning lies somewhere inbetween, and is assessed empirically in the next section. Suppose we need to approximate $u \otimes A + h \otimes D$ unbiasedly by a single Kronecker-product. For simplicity, let us assume that all vectors and matrices have norm 1. This makes the variance reduction technique of UORO and KF-RTRL, which is also implicitly included in the OK algorithm, unnecessary.

Case 1: $u = h$. Then, we can simply rewrite $u \otimes A + h \otimes D = u \otimes (A + D)$, which is a single Kronecker-product and doesn't need a noisy approximation. This shows that an optimal approximator like OK has variance 0. On the other hand, KF-RTRL performs the sign trick and approximates the sum either by $(u+h) \otimes (A+D) = 2u \otimes (A+D)$ or by $(u-h) \otimes (A-D) = 0$ and thus has variance $\|A+D\|^2$.
Case 2: $u \perp h$ and $A \perp D$. With the methods presented in the appendix, it can be shown that the sign-trick performed by KF-RTRL is optimal and that therefore there is no difference between KF-RTRL and OK in this case.

We now inspect r-OK for $r > 1$. In this case, OK takes r times more runtime and memory than KF-RTRL. To make comparisons fair, we compare OK to running r independent copies of KF-RTRL and taking their average to perform gradient descent[4], let us refer to this algorithm as r-KF-RTRL-AVG, or simply KF-AVG. Again, it can be deduced from Theorem 1 that the noise of OK is at most that of KF-AVG. To see this, observe that KF-AVG stores r Kronecker-products at each step and mixes them with $h_t \otimes D_t$ to obtain another r Kronecker-factors for the next step.
For $r > 1$ there is an additional, important phenomenon improving OK over KF-AVG, which we illustrate now. Suppose $r = 2$ and we want to approximate $u_1 \otimes A_1 + u_2 \otimes A_2 + h \otimes D$ unbiasedly by a sum of two Kronecker-products. Assume that u_1, u_2, h and A_1, A_2, D respectively are pairwise orthogonal[5] and consider the case where one of the summands is larger than the other two, say $\|u_1 \otimes A_1\| = 10$ and $\|u_2 \otimes A_2\| = \|h \otimes D\| = 1$. Then, the optimal approximator OK will keep $u_1 \otimes A_1$ fixed and mix only the other two summands creating noise of order 1. More naïve approaches, including the sign trick of KF-AVG, mixes all factors and create noise of order $10/r$, $r = 2$. This phenomenon of keeping important parts of the gradient and only mixing less important parts to reduce the noise becomes important and appears more frequently as r gets larger, see also Figure 3 for experimental evidence.

4. Experiments

Here, we empirically analyze the advantage of using the optimal approximation from OK as opposed to the sign trick from KF-RTRL. Moreover, we compare OK to TBPTT, showing that the noise in OK is so small that it does not hinder its learning performance. We posit that this is due to the noise of OK being smaller than that of Stochastic Gradient Descent (Robbins & Monro, 1951). This is independent of the batch size b, as both sources of noise are divided by the same factor b. Figure A.1 in the appendix, which is similar to Figure 2 but with larger batches, portrays this point.

[4]This reduces the noise of KF-RTRL by a factor of r.

[5]In fact, any sum can be rewritten in such a way. This is equivalent to SVD.

Table 1. Computational costs for different algorithms, measured per batch element and parameter update. These values reflect the cost in an actual implementation. The additional cost of storing the model (memory n^2) does not scale with the batch size and is therefore negligible when training with large mini-batches.
Dashed horizontal lines group algorithms with comparable costs. r is a parameter of the algorithms, T is the truncation horizon of TBPTT. See Section 3 for details.

	MEMORY	RUNTIME
RTRL	n^3	n^4
r-OK	rn^2	rn^3
r-KF-RTRL-AVG	rn^2	rn^3
UORO	n^2	n^2
r-KTP	rn	rn^2
TBPTT-T	Tn	Tn^2

Following (Mujika et al., 2018), we assess the learning performance of OK in two tasks. The first, termed Copy task, is a synthetic binary string memorization task which evaluates the RNN's ability to store information and learn long-term dependencies. The second, character-level language modeling on the Penn TreeBank dataset (CHAR-PTB), is a complex real-world task commonly used to assess the capabilities of RNNs. We compare the performance of OK to KF-RTRL and TBPTT based on 'data time', i.e. on how much data the algorithm is given. Moreover, we perform an experiment analyzing the variance of OK and KF-RTRL by comparing them to the exact gradients given by RTRL. Lastly, we measure the performance of our second algorithm KTP on the Copy task and show that it can learn moderate time dependencies. For all experiments, we use a single-layer Recurrent Highway Network (Zilly et al., 2017)[6].

4.1. Comparisons between OK, KF-RTRL and TBPTT

4.1.1. COPY TASK

For the Copy task, a binary string of length T is presented sequentially to an RNN. Once the full string is presented, the RNN should reconstruct the original string without any extra information (example for a sequence of length 5: input #01101****** and target output ******#01101). The results are shown in Figure 1. OK outperforms KF-RTRL when making a comparison that equates the memory and runtime requirements between the two approaches (see Section 3.6). Furthermore, by exploiting the online updates, OK also outperforms TBPTT when giving both algorithms the same network and batch sizes. It is important to note that the runtime and memory advantage of TBPTT imply

[6]For implementation simplicity, we replace $\tanh(x)$ by $2 * \text{sigmoid}(x) - 1$ as the non-linearity function. These functions have similar properties, so this should not have any significant effect on learning.

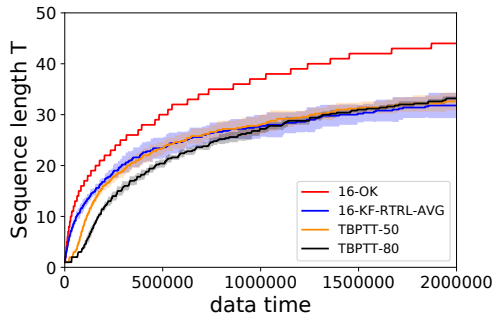

Figure 1. Copy task. We plot the mean and standard deviation (shaded area) over 5 trials. 16-OK learns sequences on average up to 42, 16-KF-RTRL-AVG up to 32, TBPTT-50 and 80 up to 33. We trained an RHN with 128 units for all models.

that it could be run on a larger network and for longer. The comparison done here is fair in the sense of giving both algorithms the same amount of data and assesses whether the noise of OK has been reduced to the point where it does not harm learning.

We now describe the details of our implementation. As in (Mujika et al., 2018), we use curriculum learning and start with $T = 1$, increasing T by one when the RNN error drops below 0.15 bits/char. After each sequence, the hidden states are reset to zero. To improve performance, the length of the sequence is sampled uniformly at random from $T - 5$ to T. This forces the network to learn a general algorithm, as opposed to one suited only for sequences of length T. We use a RHN with 128 units and a batch size of 16. We optimize the log-likelihood using the Adam optimizer (Kingma & Ba, 2015) with default Tensorflow (Abadi et al., 2016) parameters, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For each model, we pick the best learning rate from $\{10^{-2.5}, 10^{-3}, 10^{-3.5}, 10^{-4}\}$. We repeat each experiment 5 times.

4.1.2. CHAR-PTB ON THE PENN TREEBANK DATASET

For the CHAR-PTB task, the network receives a text character by character, and at each time step it must predict the next character. This is a standard, challenging test for RNNs which requires capturing long- and short-term dependencies. It is highly stochastic, as there are many potential continuations for most input sequences. Figure 2 and Table 2 show the results. 8-OK outperforms 8-KF-RTRL-AVG, and matches the performance of TBPTT-25. In fact, 8-OK even takes advantage of its online updates to achieve faster convergence when compared to TBPTT-25. The advantage observed in Figure 2 is even larger when using longer truncation horizons as suggested by Figure 1. This fact showcases the strength of performing online updates as in RTRL as opposed to having an update lock as in TBPTT.

For this experiment we use the Penn TreeBank (Marcus et al., 1993) dataset, a collection of Wall Street Journal articles commonly used for training character level models. We split the data following (Mikolov et al., 2012). In addition, we reset the hidden state to zero with a probability of 0.01 at every step (Melis et al., 2018). The experimental setup is the same as in Section 4.1.1, except the RHN has 256 units and the batch size is 32. The learning rates are chosen in the same range.

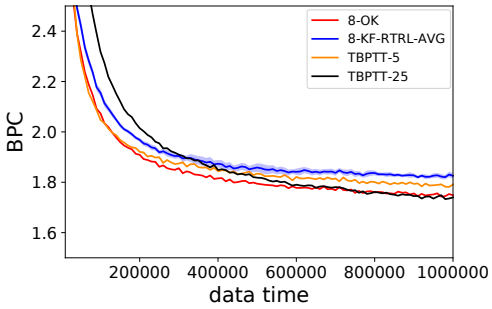

Figure 2. Validation performance on Penn TreeBank in bits per character (BPC). 8-OK matches the performance of TBPTT-25. We trained a RHN with 256 units for all models. Table 2 summarizes the performances.

Table 2. Results on Penn TreeBank. Merity et al. (2018) is the current state of the art. Standard deviations are smaller than 0.01.

NAME	VALIDATION	TEST	#PARAMS
8-KF-RTRL-AVG	1.82	1.77	$133K$
8-OK	1.74	1.69	$133K$
TBPTT-5	1.78	1.73	$133K$
TBPTT-25	1.73	1.69	$133K$
MERITY ET AL. (2018)	–	1.18	$13.8M$

4.2. Empirical Exploration of Noise

Here, we empirically evaluate how the noise evolves over time. We report the cosine between the true gradient and the approximated one. The results for an untrained network are given in the appendix (Figure A.2). There, already 2-OK achieves a cosine of almost exactly 1. However, the most interesting behavior arises later in training. Figure 3 shows that, after a million steps of training on CHAR-PTB, the cosine is much smaller for 8-KF-RTRL-AVG than for 8-OK.

For this experiment, we train a RHN with 256 units on CHAR-PTB for 1 million steps. Then, we freeze the weights of the network and compute the angle ϕ between the gradient estimates provided by OK and KF-RTRL and the true RTRL gradient for 1000 steps. We plot the mean and stan-

dard deviation of 20 repetitions of each experiment. In the appendix, Figure A.3 provides similar experiments for the Copy task.

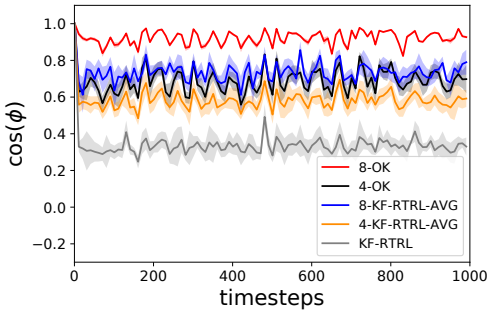

Figure 3. Variance analysis in a RHN trained for 1 million steps on CHAR-PTB. We plot the cosine of the angle between the approximated and the true value of $\frac{dL}{d\theta}$. A cosine of 1 implies that the approximation and the true value are aligned, whereas a random vector has an expected cosine of 0.

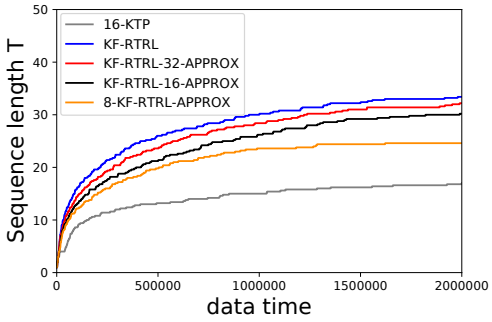

Figure 4. KTP performance on the Copy task. From top to bottom as in the legend, the learned sequence lengths are: 17, 33, 32, 30, 25. Standard deviations were around 3 for all algorithms.

4.3. Kronecker Triple Product

We now analyze the performance of KTP. While KTP has the same memory and runtime requirements as TBPTT, this comes at the cost of extra noise. KTP possesses two sources of noise (see Section 3.5). The first is added by a low-rank approximation D' of the diagonal D, the second is added in the mixing procedure. Here, we show experiments indicating that the first noise source is not significant by artificially introducing it to KF-RTRL. This is done by unbiasedly approximating $F_t = h \otimes D$ by $h \otimes D'$, where D' is as in Section 3.5. The rest of the KF-RTRL algorithm remains as usual. We term this adapted version KF-RTRL-r-APPROX when D is approximated by a rank r matrix. Figure 4 shows that KF-RTRL-16-APPROX performs almost as well as the original KF-RTRL, suggesting that the noise added in the mixing procedure is what hurts KTP the most.

For the experiment, we use the same setup as in Section 4.1.1 except that the batch sizes used were 256. We plot the mean for 5 repetitions of the experiment.

5. Conclusions

We presented two new algorithms, OK and KTP, for training RNNs. Both are unbiased approximations of Real Time Recurrent Learning (RTRL), an online alternative to Truncated Backpropagation through Time (TBPTT) giving untruncated gradients. For OK, we do not only show that it has less variance than previous approximations, but show that our approximation is in fact optimal for the class of Kronecker-Sum approximations, which includes all previously published approaches. We empirically show that this improvement makes the noise of OK negligible, which distinguishes OK from previous approximations of RTRL. This is evaluated on the standard benchmark of Penn TreeBank (PTB) character-level modeling, where OK matches the performance of TBPTT. In the case of a synthetic string-memorization task, OK can exploit frequent online parameter updates to outperform TBPTT. Our second algorithm, KTP, is more exploratory and paves the way towards more memory and runtime efficient approximations of RTRL. Its computational cost matches that of TBPTT and we show that it can learn moderate time dependencies. Reducing the noise of KTP provides an interesting problem for further research.

Our theoretical optimality result shows that, if the noise of RTRL is to be reduced further, new classes of approximations need to be explored. Moreover, the result can be extended to test the theoretical limitations of new approximations of RTRL which obey a similar structure as the Kronecker-Sum. We also presented a more general algorithm to construct unbiased, low-rank approximators of matrices with minimum achievable variance. This might be useful in other areas of machine learning which rely on unbiased gradients.

Conceptually, we explore an alternative to TBPTT. We believe that this is a necessary step towards learning long-term dependencies and for making full use of the architectural developments that have recently advanced RNNs. While RTRL itself is infeasible due to large computational costs, our results indicate that it is possible to reduce its memory and runtime requirements by a factor of n while keeping the noise small enough to not harm learning. Further improvements in this direction would already make RTRL a viable alternative to TBPTT and impact modeling data with inherent long-term dependencies.

Acknowledgements

We would like to thank Florian Meier and Pascal Su for helpful discussion and valuable comments on the presentation of this work. We also thank the anonymous reviewers for their helpful feedback.

Frederik Benzing was supported by grant no. 200021_169242 of the Swiss National Science Foundation. Marcelo Matheus Gauy was supported by CNPq grant no. 248952/2013-7. Asier Mujika was supported by grant no. CRSII5_173721 of the Swiss National Science Foundation.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Cline, A. K. and Dhillon, I. S. *Computation of the Singular Value Decomposition*. CRC Press, jan 2006.

Golub, G. H. and Van Loan, C. F. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.

Hillar, C. J. and Lim, L.-H. Most tensor problems are np-hard. *J. ACM*, 60(6):45:1–45:39, November 2013. ISSN 0004-5411. doi: 10.1145/2512329. URL http://doi.acm.org/10.1145/2512329.

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

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., and Kavukcuoglu, K. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1627–1635. JMLR. org, 2017.

Jaeger, H. The echo state approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

Ke, N. R., GOYAL, A. G. A. P., Bilaniuk, O., Binas, J., Mozer, M. C., Pal, C., and Bengio, Y. Sparse attentive backtracking: Temporal credit assignment through reminding. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 7651–7662. Curran Associates, Inc., 2018.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Liao, R., Xiong, Y., Fetaya, E., Zhang, L., Yoon, K., Pitkow, X., Urtasun, R., and Zemel, R. Reviving and improving recurrent back-propagation. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3082–3091, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/liao18c.html.

Lukoševičius, M. and Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

Maass, W., Natschläger, T., and Markram, H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. Building a large annotated corpus of english: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A. C., and Bengio, Y. Samplernn: An unconditional end-to-end neural audio generation model. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. URL https://openreview.net/forum?id=SkxKPDv5xl.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*, 2018.

Merity, S., Keskar, N. S., and Socher, R. An analysis of neural language modeling at multiple scales. *arXiv preprint arXiv:1803.08240*, 2018.

Mikolov, T., Sutskever, I., Deoras, A., Le, H.-S., Kombrink, S., and Cernocky, J. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Mujika, A., Meier, F., and Steger, A. Approximating real-time recurrent learning with random kronecker factors. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 6594–6603. Curran Associates, Inc., 2018.

Ororbia, A., Mali, A., Giles, C. L., and Kifer, D. Online learning of recurrent neural architectures by locally aligning distributed representations. *CoRR*, abs/1810.07411, 2018. URL http://arxiv.org/abs/1810.07411.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318, 2013.

Robbins, H. and Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3): 400–407, 1951.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pp. 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. URL http://dl.acm.org/citation.cfm?id=104279.104293.

Scellier, B. and Bengio, Y. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 11: 24, 2017. ISSN 1662-5188. doi: 10.3389/fncom.2017.00024. URL https://www.frontiersin.org/article/10.3389/fncom.2017.00024.

Tallec, C. and Ollivier, Y. Unbiased online recurrent optimization. *arXiv preprint arXiv:1702.05043*, 2017a.

Tallec, C. and Ollivier, Y. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017b.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Williams, R. J. and Peng, J. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2:490–501, 1990.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. Recurrent highway networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 4189–4198, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/zilly17a.html.