Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

On the vanishing and exploding gradient problem in Gated Recurrent Units

Alexander Rehmer, Andreas Kroll

Department of Measurement and Control, Institute for System Analytics and Control, Faculty of Mechanical Engineering, University of Kassel, Germany, (e-mail: {alexander.rehmer, andreas.kroll}@mrt.uni-kassel.de)

Abstract: Recurrent Neural Networks are applied in areas such as speech recognition, natural language and video processing, and the identification of nonlinear state space models. Conventional Recurrent Neural Networks, e.g. the Elman Network, are hard to train. A more recently developed class of recurrent neural networks, so-called Gated Units, outperform their counterparts on virtually every task. This paper aims to provide additional insights into the differences between RNNs and Gated Units in order to explain the superior perfomance of gated recurrent units. It is argued, that Gated Units are easier to optimize not because they solve the vanishing gradient problem, but because they circumvent the emergence of large local gradients.

Keywords: Nonlinear system identification, Recurrent Neural Networks, Gated Recurrent Units.

1. INTRODUCTION

Gated Units, such as the Long Short-Term Memory (LSTM) and the Gated Recurrent Unit (GRU) were originally developed to overcome the *vanishing gradient problem*, which occurs in the Elman Recurrent Neural Network (RNN) (Pascanu et al., 2012). They have since outperformed RNNs on a number of tasks, such as natural language, speech and video processing (Jordan et al., 2019) and recently also on a nonlinear system identification task (Rehmer and Kroll, 2019). However, it will be shown, that the gradient also vanishes in Gated Units, other still unaccounted for mechanisms have to be responsible for their success. Pascanu et al. (2012) show, that small changes in the parameters $\boldsymbol{\theta}$ of the RNN can lead to drastic changes in the dynamic behavior of the system, when crossing certain critical bifurcation points. This in turn results in a huge change in the evolution of the hidden state $\hat{\boldsymbol{x}}_k$, which leads to a locally large, or *exploding*, gradient of the loss function. In this paper the GRU will be examined and compared to the RNN with the purpose to provide an alternative explanation to why GRUs outperform RNNs. First, it will be shown, that the gradient of the GRU is in fact smaller than that of the RNN, at least for the parameterizations considered in this paper, although GRUs were originally designed to solve the vanishing gradient problem. Secondly, it will be shown, that GRUs are not only capable to represent highly nonlinear dynamics, but are also able to represent approximately linear dynamics via a number of different parameterizations. Since a linear model is always a good first guess, the easy accessibility of different parameterizations that produce linear dynamics makes the GRU less sensitive to its initial choice of parameters and thus simplifies the optimization problem. In the end GRU and RNNs will be compared on a simple academic example and on a real nonlinear identification task.

⋆ Sponsor and financial support acknowledgment goes here. Paper titles should be written in uppercase and lowercase letters, not all uppercase.

2. RECURRENT NEURAL NETWORKS

In this section the Simple Recurrent Neural Network (RNN), also known as Elman Network, and the Gated Recurrent Unit (GRU) will be introduced.

2.1 Simple Recurrent Neural Network

The RNN as depicted in figure 1 is a straightforward realization of a nonlinear state space model (Nelles, 2001). It consists of one hidden recurrent layer with nonlinear activation function $\boldsymbol{f}_\mathrm{h}$, which aims to approximate the state equation, as well as one hidden feedforward layer $\boldsymbol{f}_\mathrm{g}$ and one linear output layer, which together aim to approximate the output equation. For simplicity of notation, the linear output layer is omitted in the following equations and figures:

$$\begin{aligned} \hat{\boldsymbol{x}}_{k+1} &= \boldsymbol{f}_\mathrm{h}\left(\boldsymbol{W}_\mathrm{x}\hat{\boldsymbol{x}}_k + \boldsymbol{W}_\mathrm{u}\boldsymbol{u}_k + \boldsymbol{b}_\mathrm{h}\right), \\ \hat{\boldsymbol{y}}_k &= \boldsymbol{f}_\mathrm{g}\left(\boldsymbol{W}_\mathrm{y}\hat{\boldsymbol{x}}_k + \boldsymbol{b}_\mathrm{g}\right), \end{aligned} \tag{1}$$

with $\hat{\boldsymbol{x}}_k \in \mathbb{R}^{n\times 1}$, $\hat{\boldsymbol{y}}_k \in \mathbb{R}^{m\times 1}$, $\boldsymbol{u}_k \in \mathbb{R}^{l\times 1}$, $\boldsymbol{W}_\mathrm{x} \in \mathbb{R}^{n\times n}$, $\boldsymbol{W}_\mathrm{u} \in \mathbb{R}^{n\times l}$, $\boldsymbol{b}_\mathrm{h} \in \mathbb{R}^{n\times 1}$, $\boldsymbol{W}_\mathrm{y} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b}_\mathrm{g} \in \mathbb{R}^{m\times 1}$ and $\boldsymbol{f}_\mathrm{h} : \mathbb{R}^{n\times 1} \to \mathbb{R}^{n\times 1}$, $\boldsymbol{f}_\mathrm{g} : \mathbb{R}^{m\times 1} \to \mathbb{R}^{m\times 1}$. Usually $\tanh(\cdot)$ is employed as nonlinear activation function. When training an RNN, the recurrent model is unfolded over the whole training sequence of length N, and the gradient of the loss function $\mathcal{L}$ with respect to the model parameters $\boldsymbol{\theta}$ is calculated. As a consequence of the feedback, the gradient of the error

$$\boldsymbol{e}_k = \left(\hat{\boldsymbol{y}}_k - \boldsymbol{y}_k^\mathrm{target}\right) \tag{2}$$

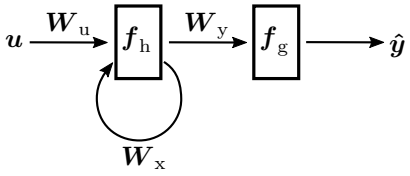


Fig. 1. Representation of the Elman Network: Layers of neurons are represented as rectangles, connections between layers represent fully connected layers.

at time step k with respect to the model parameters $\boldsymbol{\theta} = \{\boldsymbol{W}_{\mathrm{x}}, \boldsymbol{W}_{\mathrm{u}}, \boldsymbol{W}_{\mathrm{y}}, \boldsymbol{b}_{\mathrm{h}}, \boldsymbol{b}_{\mathrm{g}}\}$ depends on the previous state $\hat{\boldsymbol{x}}_{k-1}$, which depends again on the model parameters:

$$\frac{\partial \boldsymbol{e}_k}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{e}_k}{\partial \hat{\boldsymbol{y}}_k}\frac{\partial \hat{\boldsymbol{y}}_k}{\partial \hat{\boldsymbol{x}}_k}\left[\frac{\partial \hat{\boldsymbol{x}}_k}{\partial \boldsymbol{\theta}} + \frac{\partial \hat{\boldsymbol{x}}_k}{\partial \hat{\boldsymbol{x}}_{k-1}}\frac{\partial \hat{\boldsymbol{x}}_{k-1}}{\partial \boldsymbol{\theta}}\right] \tag{3}$$

For example, the gradient of the hidden state $\hat{\boldsymbol{x}}_k$ with respect to $\boldsymbol{W}_{\mathrm{x}}$ is

$$\frac{\partial \hat{\boldsymbol{x}}_k}{\partial \boldsymbol{W}_{\mathrm{x}}} = \sum_{\tau=1}^{N} \hat{\boldsymbol{x}}_{k-\tau} \boldsymbol{f}_{\mathrm{h}}'^{(k-\tau+1)}(\cdot) \prod_{\beta} \boldsymbol{f}_{\mathrm{h}}'^{(k-\beta)}(\cdot)\, \boldsymbol{W}_{\mathrm{x}}^{(k-\beta)} \tag{4}$$
$$\beta = \tau - 2,\ \tau - 3,\ \ldots \forall \beta \geq 0.$$

High indices in brackets indicate the particular time step. The product term in (4), which also appears when computing the gradient with respect to the other parameters, decreases exponentially with τ, if $|f_{\mathrm{h}}' \cdot \rho(\boldsymbol{W})| < 1$, where $\rho(\boldsymbol{W}_{\mathrm{x}})$ is the spectral radius of $\boldsymbol{W}_{\mathrm{x}}$. Essentially, backpropagating an error one time step involves a multiplication of the state with a derivative that is possibly smaller than one and a matrix whose spectral radius is possibly smaller than one. Hence, the gradient vanishes after a certain amount of time steps. In the Machine Learning community it is argued that the vanishing gradient prevents learning of so-called *long-term dependencies* in acceptable time (Hochreiter and Schmidhuber, 1997; Goodfellow et al., 2016), i.e. when huge time lags exist between input $\boldsymbol{u}_k$ and output $\hat{\boldsymbol{y}}_k$. Gated recurrent units, like LSTM and GRU were developed to solve this problem and have since then outperformed classical RNNs on virtually any task. However, it can be shown that the gradient also vanishes in gated recurrent units. Additionally, the vanishing of the gradient over time is a desirable property. In most systems, the influence of a previous state $\boldsymbol{x}_{k-\tau}$ on the current state $\hat{\boldsymbol{x}}_k$ decreases over time. Unless one wants to design a marginally stable or unstable system, e.g. when performing tasks like unbounded counting, or when dealing with large dead times, the vanishing gradient has no negative effect on the optimization procedure.

2.2 The Gated Recurrent Unit

Gated Recurrent Unit (GRU) (Cho et al., 2014) is besides LSTM the most often applied architecture of gated recurrent units. The general concept of gated recurrent units is to manipulate the state $\hat{\boldsymbol{x}}_k$ through the addition or multiplication of the activations of so called gates, see figure 2. Gates are almost exclusively one-layered neural networks with nonlinear sigmoid activation functions. The state equation of the GRU is

$$\hat{\boldsymbol{x}}_{k+1} = \boldsymbol{f}_{\mathrm{z}} \odot \hat{\boldsymbol{x}}_k + (\boldsymbol{1} - \boldsymbol{f}_{\mathrm{z}}) \odot \boldsymbol{f}_{\mathrm{c}}(\tilde{\boldsymbol{x}}_k) \tag{5}$$

with $\tilde{\boldsymbol{x}}_k = \boldsymbol{f}_{\mathrm{r}} \odot \hat{\boldsymbol{x}}_k$. The operator $\odot$ denotes the Hadamard product. The activations of the so-called gate reset gate $\boldsymbol{f}_{\mathrm{r}}$, update gate $\boldsymbol{f}_{\mathrm{z}}$ and the output gate $\boldsymbol{f}_{\mathrm{c}}$ are given by

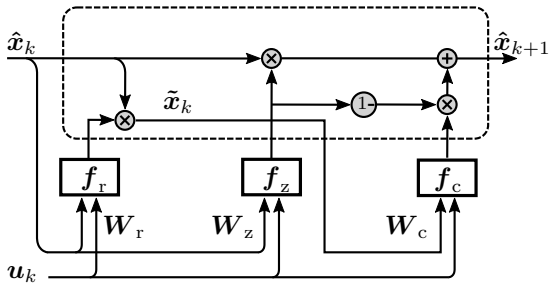


Fig. 2. The Gated Recurrent Unit (GRU). Gates are depicted as rectangles with their respective activation functions.

$$\begin{aligned}
\boldsymbol{f}_{\mathrm{r}} &= \boldsymbol{\sigma}\left(\boldsymbol{W}_{\mathrm{r}} \cdot [\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k] + \boldsymbol{b}_{\mathrm{r}}\right), \\
\boldsymbol{f}_{\mathrm{z}} &= \boldsymbol{\sigma}\left(\boldsymbol{W}_{\mathrm{z}} \cdot [\hat{\boldsymbol{x}}_k, \boldsymbol{u}_k] + \boldsymbol{b}_{\mathrm{z}}\right), \\
\boldsymbol{f}_{\mathrm{c}} &= \boldsymbol{\tanh}\left(\boldsymbol{W}_{\mathrm{c}} \cdot [\tilde{\boldsymbol{x}}_k, \boldsymbol{u}_k] + \boldsymbol{b}_{\mathrm{c}}\right),
\end{aligned} \tag{6}$$

where $\boldsymbol{W}_{\mathrm{r}}, \boldsymbol{W}_{\mathrm{z}}, \boldsymbol{W}_{\mathrm{c}} \in \mathbb{R}^{n \times n+l}$, $\boldsymbol{b}_{\mathrm{r}}, \boldsymbol{b}_{\mathrm{z}}, \boldsymbol{b}_{\mathrm{c}} \in \mathbb{R}^{n \times 1}$ and $\boldsymbol{f}_{\mathrm{r}}, \boldsymbol{f}_{\mathrm{z}}, \boldsymbol{f}_{\mathrm{c}} : \mathbb{R}^{n \times 1} \to \mathbb{R}^{n \times 1}$. $\boldsymbol{\sigma}(\cdot)$ denotes the logistic function.In order to map the states estimated by the GRU to the output, the GRU has to be equipped either with an output layer, as the RNN, or with an output gate, as the LSTM.

3. GRADIENT OF THE STATE EQUATIONS

In this section, the gradients of the state equations of RNN (1) and GRU (5) w.r.t. their parameters will be compared to each other. In the cases examined the gradient of the GRU is, somewhat surprisingly, at most as large as that of the RNN, but usually smaller.
In order to allow for an easily interpretable visualization, the analysis will be restricted to one dimensional and autonomous systems. Also, the GRU will be simplified by eliminating the reset gate $\boldsymbol{f}_{\mathrm{r}}$ from (6), such that $\tilde{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k$. Taking the gradient of the RNN's state equation in (1) w.r.t. w_{x} yields

$$\frac{\partial \hat{x}_{k+1}}{\partial w_{\mathrm{x}}} = (\hat{x}_k + w_{\mathrm{x}}\frac{\partial \hat{x}_k}{\partial w_{\mathrm{x}}}) \cdot \tanh'(w_{\mathrm{x}}\hat{x}_k + b_{\mathrm{x}}). \tag{7}$$

The gradient of the GRUs state equation (5) with respect to w_{z} is

$$\begin{aligned}
\frac{\partial \hat{x}_{k+1}}{\partial w_{\mathrm{z}}} &= (1 - \tanh(\hat{x}_k; \boldsymbol{\theta}_{\mathrm{c}}))\, \sigma'(\hat{x}_k; \boldsymbol{\theta}_{\mathrm{z}}) \left(\hat{x}_k + w_{\mathrm{z}}\frac{\partial \hat{x}_k}{\partial w_{\mathrm{z}}}\right) \\
&\quad + (1 - \sigma(\hat{x}_k; \boldsymbol{\theta}_{\mathrm{z}}))\tanh'(\hat{x}_k; \boldsymbol{\theta}_{\mathrm{c}})\frac{\partial \hat{x}_k}{\partial w_{\mathrm{z}}},
\end{aligned} \tag{8}$$

and with respect to w_{c}

$$\begin{aligned}
\frac{\partial \hat{x}_{k+1}}{\partial w_{\mathrm{c}}} &= \sigma'(\hat{x}_k; \boldsymbol{\theta}_{\mathrm{z}})\frac{\partial \hat{x}_k}{\partial w_{\mathrm{c}}}(\hat{x}_k - \tanh(\hat{x}_k; \boldsymbol{\theta}_{\mathrm{c}})) \\
&\quad + \sigma(\hat{x}_k; \boldsymbol{\theta}_{\mathrm{z}})\frac{\partial \hat{x}_k}{\partial w_{\mathrm{c}}} \\
&\quad + (1 - \sigma(\hat{x}_k; \boldsymbol{\theta}_{\mathrm{z}}))\tanh'(\hat{x}_k; \boldsymbol{\theta}_{\mathrm{c}})\left(\hat{x}_k + w_{\mathrm{c}}\frac{\partial \hat{x}_k}{\partial w_{\mathrm{c}}}\right).
\end{aligned} \tag{9}$$

For convenience of notation $\boldsymbol{\theta}_{\mathrm{z}}$ and $\boldsymbol{\theta}_{\mathrm{c}}$ denote the parameters of f_{z} and f_{c} respectively, i.e. $\boldsymbol{\theta}_{\mathrm{z}} = [w_{\mathrm{z}}, b_{\mathrm{z}}]$ and $\boldsymbol{\theta}_{\mathrm{c}} = [w_{\mathrm{c}}, b_{\mathrm{c}}]$.
It is cumbersome to write down the gradients in (8) and (9) for an arbitrary number of time steps, as done for the

RNN in (4). However, it already becomes apparent from (8) and (9), that each step of backpropagation involves the multiplication of the previous state with the output of a sigmoid function, either $\sigma(\cdot)$ or $\tanh(\cdot)$, and with one of its derivatives. The outputs of the sigmoid functions is always smaller than one. The derivative of the $tanh(\cdot)$ is smaller than one, if $|\boldsymbol{W}_\mathrm{c}|_1 < 1$ and the derivative of the logistic function is smaller than one, if $|\boldsymbol{W}_\mathrm{z}|_1 < 4$. This means, the gradient of the GRU is at least as likely to vanish as the gradient of the RNN.

In order to examine the gradients further it is assumed, that $|w_\mathrm{x}|, |w_\mathrm{z}|, |w_\mathrm{c}| \leq 1$. The derivative of the logistic function $\sigma(\cdot)$ is therefore bounded in the interval $\left[0, \frac{1}{4}\right]$ and the derivative of the $\tanh(\cdot)$ bounded in the interval $[0, 1]$. Under these assumptions the gradient of the RNN is at most

$$\frac{\partial \hat{x}_{k+1}}{\partial w_\mathrm{x}} \leq (\hat{x}_k + w_\mathrm{x} \frac{\partial \hat{x}_k}{\partial w_\mathrm{x}}). \tag{10}$$

For the gradient of the GRU two extreme cases will be examined and compared to (10).

If the update gate is fully closed, i.e. $\sigma(\hat{x}_k; \boldsymbol{\theta}_\mathrm{z}) = 0$, (8) and (9) become

$$\begin{aligned} \frac{\partial \hat{x}_{k+1}}{\partial w_\mathrm{z}} &= \tanh'(\hat{x}_k; \boldsymbol{\theta}_\mathrm{c}) \frac{\partial \hat{x}_k}{\partial w_\mathrm{z}} \leq \frac{\partial \hat{x}_k}{\partial w_\mathrm{z}} \\ \frac{\partial \hat{x}_{k+1}}{\partial w_\mathrm{c}} &= \tanh'(\hat{x}_k; \boldsymbol{\theta}_\mathrm{c}) (\hat{x}_k + w_\mathrm{c} \frac{\partial \hat{x}_k}{\partial w_\mathrm{c}}) \\ &\leq \hat{x}_k + w_\mathrm{c} \frac{\partial \hat{x}_k}{\partial w_\mathrm{c}} \end{aligned} \tag{11}$$

It follows, that if the update gate is fully closed, the GRU becomes an ordinary RNN with the same properties regarding its gradient.

If the update gate is fully open on the other hand, i.e. $\sigma(\hat{x}_k; \boldsymbol{\theta}_\mathrm{z}) = 1$, (8) and (9) become

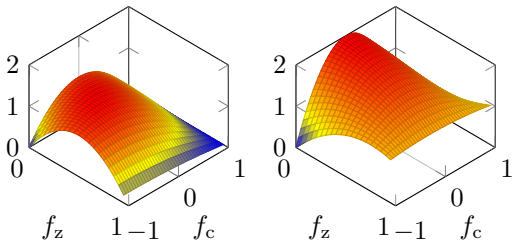
$$\begin{aligned} \frac{\partial \hat{x}_{k+1}}{\partial w_\mathrm{z}} &= 0, \\ \frac{\partial \hat{x}_{k+1}}{\partial w_\mathrm{c}} &= \frac{\partial \hat{x}_k}{\partial w_\mathrm{c}}. \end{aligned} \tag{12}$$

Essentially, the previous state $\hat{x}_k$ is just passed over to the new state $\hat{x}_{k+1}$ without any modification, so there is no gradient w.r.t. the parameters but that from the previous time-step.

One can examine the behavior of (8) and (9) in between these extreme cases by replacing $\tanh'(\cdot) = 1 - \tanh^2(\cdot)$ and $\sigma'(\cdot) = \sigma(\cdot)(1 - \sigma(\cdot))$. (8) and (9) then each are a cubic polynomial in $\sigma(\hat{x}_k, \theta_z)$ and $\tanh(\hat{x}_k, \theta_c)$. Their graphs are plotted in figure 3 for $w_\mathrm{z} = w_\mathrm{c} = 1$, which produces the largest gradient possible under the assumptions made. Also we set $\frac{\partial \hat{\boldsymbol{x}}_k}{\partial w_\mathrm{z}} = \frac{\partial \hat{\boldsymbol{x}}_k}{\partial w_\mathrm{c}} = 1$ and $\hat{x}_k = 1$.

Figure 3 confirms, that the largest gradients occurs, when the GRU's update gate is fully closed, i.e. $\sigma(\hat{x}_k, \theta_\mathrm{z}) = 0$, and the GRU hence becomes an RNN. For all other configurations, the gradient is in fact smaller.

In contrast to the point made by the *vanishing gradient* argument, that if the gradient vanishes after some time-steps, optimization might take prohibitively long. We argue that a smaller gradient of the state equation w.r.t. its parameters is beneficial when training recurrent neural networks. If the gradient is large, as is the case with the RNN, a small change in parameters will lead to a huge change in the state trajectory. Since every time step, the previous state is again fed to the recurrent network, these



(a) $\partial \hat{x}_{k+1}/\partial w_\mathrm{z}$ (b) $\partial \hat{x}_{k+1}/\partial w_\mathrm{c}$

Fig. 3. Gradients of the GRU w.r.t. w_z and w_c depending on the activation of $f_\mathrm{z} = \sigma(\hat{x}_k, \theta_\mathrm{z})$ and $f_\mathrm{c} = \tanh(\hat{x}_k, \theta_\mathrm{c})$ for $w_\mathrm{z} = w_\mathrm{c} = \frac{\partial \hat{\boldsymbol{x}}_k}{\partial w_\mathrm{z}} = \frac{\partial \hat{\boldsymbol{x}}_k}{\partial w_\mathrm{c}} = \hat{x}_k = 1$

changes accumulate quickly, resulting in vastly different evolutions of the state. The effect on the loss function are locally huge gradients, which make gradient-based optimization very difficult. Compared to the RNN, the GRUs gradient of the state equation is almost always smaller, which means similar parameters produce similar trajectories of the state, leading to a smoother loss function without huge local gradients. This will be illustrated via simple examples in section 5.

4. EFFECT OF DIFFERENT PARAMETERIZATIONS ON THE STATE EQUATION

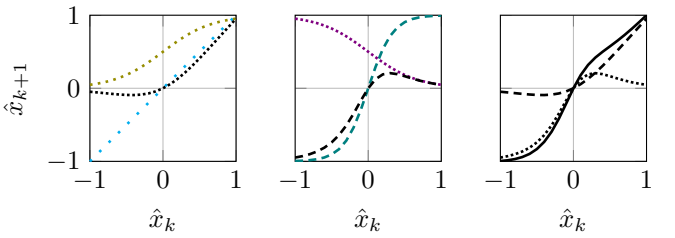
In addition to the smaller gradient of the GRU, the rather special structure of its state equation also benefits gradient based optimization, especially when identifying technical systems.

In this section, the same assumptions and restrictions apply as in the previous section, i.e. only one dimensional and autonomous systems are investigated and the GRUs reset gate $\boldsymbol{f}_\mathrm{r}$ is neglected.

Without reset gate, (5) is the sum of the identity function, weighted with a logistic function $\sigma(\cdot)$, and a $\tanh(\cdot)$, weighted by the residual $1 - \sigma(\cdot)$. Fig. 4 visually decomposes (5) into its constituent parts.

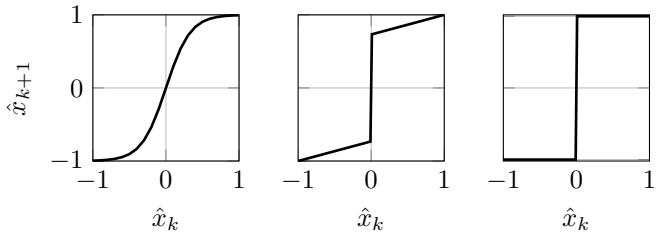
This superposition of multiple functions enables the GRU to approximate quite complex nonlinearities with comparatively few parameters.

For example, if $b_\mathrm{z} \to -\infty$, (5) converges to a tanh parameterized by w_c and b_c. For $w_\mathrm{z}, b_\mathrm{c} \to 0$, $w_\mathrm{c} \to +\infty$ and $b_\mathrm{z} \to -\infty$, (5) converges to the binary step function, while for small negative b_z it represents a kind of leaky binary step, where the slope of the tails is $\tilde{b}_z$, compare Figure 5:



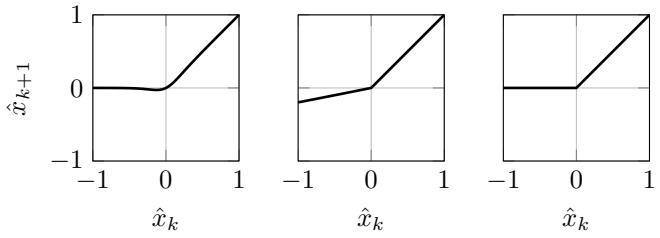
(a) $f_\mathrm{z} \cdot \hat{x}_k$ (·····) (b) $(1 - f_\mathrm{z}) f_\mathrm{c}$ (- - -) (c) $f_\mathrm{z} \cdot \hat{x}_k + (1 - f_\mathrm{z}) \cdot f_\mathrm{c}$ (——)

Fig. 4. Graphical decomposition of the GRU's state equation. $\hat{x}_k$ (· · · ·), f_z (·····), f_c (- - -), $(1 - f_\mathrm{z})$ (·····)



(a) $b_z \to -\infty$ (b) $w_z, w_c \to 0$ (c) $w_z, b_c \to 0$, $w_c \to \infty$, $b_z \to -\infty$

Fig. 5. Parameterizations for which GRU converges towards a tanh, leaky binary step or binary step activation function.



(a) $w_c, b_z, b_c \to 0$ (b) $b_z \to +\infty$, $w_z, b_c \to 0$ (c) $b_z \to \infty$, $w_z, w_c, b_c \to 0$

Fig. 6. Parameterizations for which GRU converges to variants of the ReLu activation function.

$$\tilde{b}_z = \sigma(b_z)$$

(5) also converges to the ReLu activation function for $w_z \to \pm\infty$ and $w_c, b_z, b_c \to \pm 0$ and is also able to represent a leaky ReLu with tail slope of $\tilde{w}_c$:

$$\tilde{w}_c = \tanh(w_c)$$

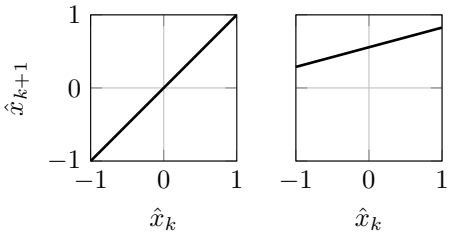
for small w_c and the swish activation for small w_z. These configurations are depicted in Figure 6.

The fact that the GRUs state equation converges to some of these functions when certain parameters become larger has an important effect on the loss function: Assume one tries to learn a state space equation in form of a tanh. Although the RNN would be the right choice, because its in the right model class, the optimization problem is rather difficult. The only solution lies in a steep valley, which means small deviations from that solution produce large losses. It would take infinitely small steps (as soon as you are in the valley, which one cannot know) to reach that solution. The GRU on the other hand offers an infinite amount of solutions for large b_z, which corresponds to an infinite plane with small slope in the parameter space. One only has to move along that plane to approximate the system increasingly better, which is easier than diving into a steep valley.

Additionally, the GRU is able to approximate the identity function and linear functions in general via a number of different parameterizations.: For $b_z \to +\infty$ (5) converges to the identity function, since f_z is approaching one. For $w_z, w_c \to 0$ the result is also a linear function, but its slope is determined by $\tilde{b}_z$ and its axis intercept by $\tilde{b}_c$, see Figure 6:

$$\tilde{b}_c = \tanh(b_c)$$

The ability to converge to a linear model when approaching different limits in the parameter space is especially important when identifying technical systems. Even for highly nonlinear processes the one step prediction surface becomes more and more linear with increasing sampling rate (Nelles, 2001). Therefore even when estimating a model for a nonlinear process, it is important that the model has the capability to represent a linear dynamical system. The fact that (5) represents a linear model for an infinite number of parameter combinations and even when approaching different limits, means, that there are huge spaces in the parameter space, which already represent a formidable solution. Representing the identity function with a one dimensional Elman Network is only possible by letting w_x and b_x approach zero, which corresponds to only a single point in the parameter space.



(a) $b_z \to +\infty$ (b) $w_z, w_c \to 0$

Fig. 7. Parameterizations for which GRU converges to linear activation functions.

5. ACADEMIC EXAMPLE

Based on the idea, that representing an approximately linear system is important even when identifying a nonlinear system, the loss functions and their gradients produced by the RNN and the GRU will be examined on the following linear discrete-time autonomous system with one state:

$$x_{k+1} = -0.9 \cdot x_k \tag{13}$$

The initial state is $x_0 = 1$ and the system is simulated for $N = 100$ time steps. The RNN is equipped with one neuron in the recurrent layer without an output layer:

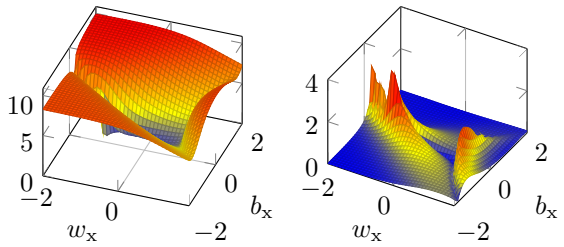
$$\hat{x}_{k+1} = \tanh(w_x \hat{x}_k + b_x) \tag{14}$$

Since the GRU has two gates with respectively two parameters, the parameters of the update gate $\boldsymbol{f}_z$ will be fixed to the reasonable values $w_z = 1$ and $b_z = 0$ when examining the loss function and gradient with respect to the output gates parameters. Conversely the parameters of the output gate $\boldsymbol{f}_c$ are fixed to the values $w_z = 1$ and $b_z = 0$ when examining the update gate.

$$\begin{aligned}\hat{x}_{k+1} = &[\sigma(w_z \hat{x}_k + b_z) \odot \hat{x}_k \\ &+ [1 - \sigma(w_z \hat{x}_k + b_z)] \cdot \tanh(w_c \hat{x}_k + b_c)]\end{aligned} \tag{15}$$

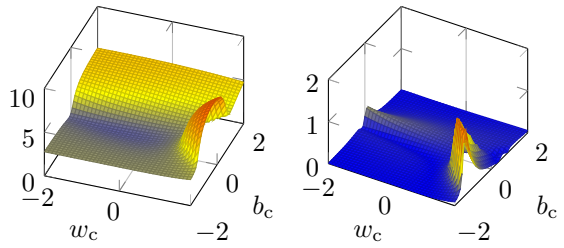
Figures 8 and 9 show the resulting loss functions of the GRU and the RNN as well as their respective gradients.

It is apparent, that the optimization problem posed by the RNN is rather difficult to solve with gradient based optimization techniques: The local optimum lies in a narrow valley surrounded by large gradients. The loss function of the GRU on the other hand does not possess such a narrow value, compare figure 9a. The reason for this is that the update gate f_z dominates the behavior of the model for $\hat{x} > 0$ (which is where training date is

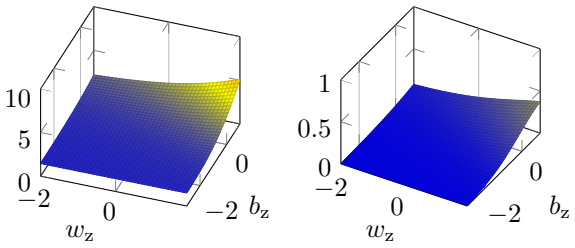


(a) RNN loss function (b) RNN gradient

Fig. 8. RNNs loss function and magnitude of the gradient on the linear identification task



(a) GRU loss function for $w_z = w_z = 1, b_z = 0$ (b) GRU gradient for $w_z = 1, b_z = 0$



(c) GRU loss function for $w_c = w_c = 1, b_c = 0$ (d) GRU gradient for $w_c = 1, b_c = 0$

Fig. 9. GRUs loss function and magnitude of the gradient on the linear identification task

available in this case study) by multiplying the output of the output gate f_c with $1 - f_z$ and thereby diminishing its influence. The effect is a smooth loss function without large gradients.

6. CASE STUDY: ELECTRO-MECHANICAL THROTTLE

To test, whether the properties of the GRU also proof beneficial in real-life applications, it was compared to an RNN on a real nonlinear dynamical system.

6.1 The test system

The system to be identified is an industrial electro-mechanical throttle, as they are employed in combustion engines, which is operated load-free in a laboratory test stand as depicted in figure 10. The input signal u is the duty cycle for the pulse width modulator (PWM). The output signal y is the voltage of the sensor measuring the angle of the throttle plate. The throttle is actuated by a DC motor. Even though the setup is relatively simple, the modeling task is rather difficult due to the characteristics

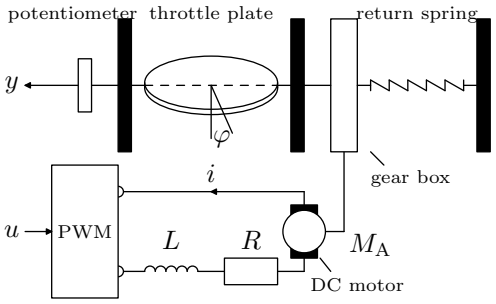


Fig. 10. Technology schematic of the electro-mechanical throttle.

of the system (Gringard and Kroll, 2016): a lower and an upper hard stop, state-dependent friction, and a nonlinear return spring.

6.2 Excitation Signals

One multisine signal and two Amplitude Modulated Pseudo-Random Binary Sequences (APRBS 1 and APRBS 2) have been used to excite the system. The multisine signal has a length of ≈ 10 s or 10^3 instances, and the APRBS signals have a length of ≈ 25 s or 2500 instances each. For the multisine signal, an upper frequency of $f_u = 7.5$ Hz has been used. For the APRBS signals, the holding time is $T_H = 0.1$ s. See (Gringard and Kroll, 2016) for more information on the test signal design.

6.3 Data preprocessing

APRBS 1 and its response signal were scaled to the interval $[-1, 1]$; all other signals were scaled accordingly. The data was then divided into training, validation and test datasets in the following way:

- Training dataset: Consists of two batches. The first batch comprises 80 % of all instances of the multisine signal and the corresponding system response. The second batch consists of 70 % of APRBS 1 and its corresponding response signal.
- Validation dataset: Consists of two batches. The first batch comprises the remaining 20 % of the multisine signal and the corresponding system response. The second batch consists of the remaining 30 % of APRBS 1 and the response signal.
- Test dataset: One batch. APRBS 2 and its corresponding system response.

This division was chosen because the multisines response signal almost exclusively covers the medium operating range while the APRBS' response signal also covers the lower and upper hard stops.

6.4 Model Architectures

The different model architectures used for identification are listed in TABLE 1. Each model architecture was initialized randomly 20 times and trained until convergence in order to examine sensitivity to initial parameters. The number of states $\dim(\boldsymbol{x})$ was varied from three to ten. Model architectures with an equal number of states were

designed to have the same number of hidden neurons in the output layer. Due to to its many gates a GRU will have significantly more parameters than an RNN with an equal number of states. This should be considered when comparing both architectures.

Table 1: Network architectures

		GRU					
1st Layer	Gated Unit	GRU					
	dim($\boldsymbol{x}$)	3	4	5	6	8	10
2nd Layer	$f_{\mathrm{g}}(\cdot)$	tanh					
	#(Neurons)	4	5	6	7	8	10
dim(θ)		66	103	149	201	321	481
		RNN					
1st Layer	$f_{\mathrm{h}}(\cdot)$	tanh					
	dim($\boldsymbol{x}$)	3	4	5	6	8	10
2nd Layer	$f_{\mathrm{g}}(\cdot)$	tanh					
	#(Neurons)	4	5	6		7	
dim(θ)		36	55	78	105	151	205

6.5 Model Training

Each of the models in Table 1 was initialized randomly 20 times and trained for 800 epochs. Between each batch, the models initial states were set to zero. Parameters were estimated based on the training dataset using the ADAM optimizer (Kingma and Ba, 2015) with its default parameter configuration ($\alpha = 0.01, \beta_1 = 0.9, \beta_2 = 0.999$).

6.6 Results

At the end of the optimization procedure, the model with the parameter configuration, which performed best on the validation dataset, was selected and evaluated on the test dataset. The performance of the models is measured in terms of their best fit rate (BFR):

$$\mathrm{BFR} = 100\% \cdot \max\left(1 - \frac{\|y_k - \hat{y}_k\|_2}{\|y_k - \bar{y}\|_2}, 0\right) \qquad (16)$$

Figure 11 shows the BFR of the RNN and the GRU on the test dataset. As expected, nonlinear optimization of the GRU consistently yields high performing models, while the performance of the RNNs fluctuates strongly. It should be noted, that there are cases where the RNNs performance matches or even exceeds the performance of the GRU (e.g. for dim($\boldsymbol{x}$) = 10). This proves, that the RNN is in general able to represent the test system, it seems just very unlikely to arrive at such a parameterization during the optimization process. Arguably because of the issues discussed in Section 3.

7. CONCLUSIONS & OUTLOOK

It was shown, that the gradient of the GRU's state equation w.r.t. its parameters is at most as large as but usually smaller than that of the RNN, provided the L_1 norm of all weights is smaller or equal to one. This finding contradicts the argument, that a vanishing gradient is responsible for the RNNs poor performance on various tasks. The first point made in this paper is, that the

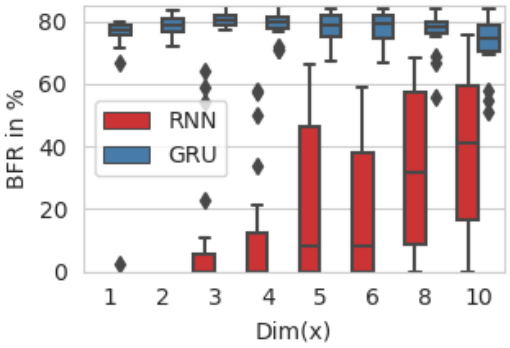


Fig. 11. Boxplot of the BFR of RNN and GRU on the test dataset. Each model was initialized 20 times and trained for 800 epochs.

smaller gradient produced by the GRU helps gradient based optimization, since small changes in the parameter space correspond to small changes in the evolution of the state, which in turn produces a smooth loss function without large gradients. The second argument made is, that the GRU's state equation converges to different functions, when certain parameters become larger. This corresponds to producing large planes in the loss function along which the solution improves steadily, rather than narrow valleys, as they are produced by the RNN. The analyses provided in this paper have yet to be generalized to state space networks with arbitrary dimensions and the whole parameter space.

REFERENCES

Cho, K. et al. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 8, 1724–1734.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gringard, M. and Kroll, A. (2016). On the systematic analysis of the impact of the parametrization of standard test signals. In *IEEE Symposium Series of Computational Intelligence 2016*. IEEE, Athens, Greece.

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

Jordan, I.D., Sokol, P.A., and Park, I.M. (2019). Gated recurrent units viewed through the lens of continuous time dynamical systems. *arXiv preprint arXiv:1906.01005*.

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations (ICLR 2015)*.

Nelles, O. (2001). *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Springer, Berlin Heidelberg, Germany.

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. *CoRR*, abs/1211.5063.

Rehmer, A. and Kroll, A. (2019). On using gated recurrent units for nonlinear system identification. In *Preprints of the 18th European Control Conference (ECC)*, 2504–2509. IFAC, Naples, Italy.