Sampling-based Gradient Regularization for Capturing Long-Term Dependencies in Recurrent Neural Networks

Artem Chernodub∗†, Dimitri Nowicki∗‡
∗Institute of Mathematical Machines and Systems of NAS of Ukraine.
†a.chernodub@gmail.com
‡nowicki@nnteam.ogr.ua

Abstract—Vanishing (and exploding) gradients effect is a common problem for recurrent neural networks with nonlinear activation functions which use backpropagation method for calculation of derivatives. Deep feedforward neural networks with many hidden layers also suffer from this effect. In this paper we propose a novel universal technique that makes the norm of the gradient stay in the suitable range. We construct a way to estimate a contribution of each training example to the norm of the long-term components of the target functions gradient. Using this subroutine we can construct mini-batches for the stochastic gradient descent (SGD) training that leads to high performance and accuracy of the trained network even for very complex tasks. We provide a straightforward mathematical estimation of minibatch’s impact on for the gradient norm and prove its correctness theoretically. To check our framework experimentally we use some special synthetic benchmarks for testing RNNs on ability to capture long-term dependencies. Our network can detect links between events in the (temporal) sequence at the range 100 and longer.

Index Terms—simple recurrent networks, gradient vanishing, regularization

I. INTRODUCTION

Recurrent Neural Networks (RNNs) are known as universal approximators of dynamic systems [1], [2]. Since RNNs are able to simulate any open dynamical system, they have a broad spectrum of applications such as time series forecasting [3], control of plants [4], language modeling [5], speech recognition [6], neural machine translation [7] and other domains. The easiest way to create an RNN is adding the feedback connections to the hidden layer of multilayer perceptron. This architecture is known as Simple Recurrent Network (SRN). Despite of the simplicity, it has rich dynamical approximation capabilities mentioned above. However, in practice training of SRNs using first-order optimization methods is difficult [8]. The main problem is well-known vanishing/exploding gradients effect that prevents capturing of long-term dependencies in data. Vanishing gradients effect is a common problem for recurrent neural networks with sigmoid-like activation functions which uses a backpropagation method for calculation of derivatives. Deep feedforward neural networks with many hidden layers also feels this effect. Hochreiter and Schmidhuber designed a set of special synthetic benchmarks for testing RNNs on ability to capture long-term dependencies [9]. They showed that ordinary SRNs are very ineffective to learn correlations in sequential data if distance between the target events is more than 10 time steps. The solution could be using more advanced second-order optimization algorithms such as Extended Kalman Filter [10], LBFGS, Hessian-Free optimization [11], but they require much more memory and computational resources for state-of-the-art networks. Another and the most popular solution is designing a neural networks architecture with more suitable dynamics. Echo State Networks (ESNs) proposed by Jaeger [12] may be considered as big reservoirs of sparsely connected neurons and randomly initialized weights which produces chaotic dynamics. For ESNs the gradients are computed for the last non-recurrent weights layer only. Experiments show that this may be enough for capturing long-term dynamics [13]. At the same time, ESNs often seems to have abundant number of free parameters. We also mention such an alternative to temporal neural networks as hierarchical sequence processing with auto-associative memories [14] that use distributed coding.

Another approach that was specially designed for catching the long-term dependencies is Long-Short Term Memory (LSTM) [9]. These neural networks are designed to adaptively reset or update their memory. They have specially designed complex structure that includes input and forgetting gates and they have constant error flow carousel. Currently it is probably the most popular family of RNNs models that shows state-of-the-art performance in several domains including speech recognition [6], image captioning [15] and neural machine translation. An idea of using input/forgetting gates inspired a lot of followers, Gated Recurrent Units (GRU) networks is probably one of the most successful of them [7]. Finally, the united team from Google and Facebook performed a grand experiment on finding the best architecture for RNNs [16]. They numerically evaluated 10,000 random architectures with 230,000 hyperparameter configurations in total and obtained a couple of new advanced models and some recommendations for easy improving of standard LSTMs. At the same time, a question how to train SRNs for catching the long-term dependencies is still a topic of current interest. It is highly desirable at least for better understanding of underlying processes of the training inside the recurrent and deep neural processes.
networks. Also, SRNs are more compact and fast working models of RNNs in comparison with ESNs and LSTMs that is very important for implementation to mobile and embedded devices. Recent research shows the ability to train SRNs for long term dependencies up to 100 time steps and more using several new techniques [17], [18]–[20]. It includes very accurate initialization of start weights, scaling down the big gradients, using advanced Nesterov momentum. One of the most common methods for preventing the vanishing gradients effect from this pool is known as “gradient regularization” [20], [21]; also it was independently proposed in [22] as “method of pseudoregularization”. The idea of gradient regularization is adding to the target cost function an additional term that controls the norm of the backpropagated gradients. In this way, neural network learns not only to fit the train data but also to keep the flow of gradients into the certain range. In this paper we propose a new method to perform the gradient regularization by selection of proper samples in dataset using the simple criteria.

II. BACKPROPAGATION MECHANISM REVISITED

Consider a Simple Recurrent Neural Network (Fig. 1) that at each time step $k$ receives an external input $u(k)$, previous internal state $z(k − 1)$ and produces output $y(k + 1)$:

$$
\begin{align*}
\mathbf{a}(k) &= \mathbf{u}(k)\mathbf{w}_{in} + \mathbf{z}(k − 1)\mathbf{w}_{rec} + \mathbf{b}, \\
\mathbf{z}(k) &= f(\mathbf{a}(k)), \\
\mathbf{y}(k + 1) &= g(\mathbf{z}(k)\mathbf{w}_{out}),
\end{align*}
$$

(1)

where $\mathbf{w}_{in}$ is a matrix of input weights, $\mathbf{w}_{rec}$ is matrix of recurrent weights, $\mathbf{w}_{out}$ is matrix of output weights, $\mathbf{a}(k)$ is known as “presynaptic activations”, $\mathbf{z}(k)$ is a network’s state, $f(\cdot)$ and $g(\cdot)$ are nonlinear activation functions for hidden and output layer respectively. In this work we always use $\tanh$ function for hidden layer and optionally $\text{softmax}$ or $\text{linear}$ function depending on the target problem (classification or regression). Also in our equations data vectors $\mathbf{u}(k)$ are vector rows (not vector columns) because it is often implemented in many software tools like MATLAB and therefore is more useful for practice.

![Fig. 1: Simple Recurrent Network](image)

After processing an input sequence $\mathbf{u}(1), \mathbf{u}(1) \ldots, \mathbf{u}(k)$ and producing the output $\mathbf{y}(k + 1)$, target error $E(k + 1)$ is calculated. There are different options for target error function, the most popular choices are mean squared error for regression and cross-entropy error for classification. To train neural network using gradient-based optimization algorithm we have to calculate derivatives of error function subs to network’s weights $\frac{\partial E}{\partial \mathbf{w}_{in}}, \frac{\partial E}{\partial \mathbf{w}_{rec}}, \frac{\partial E}{\partial \mathbf{w}_{out}}$. To perform this, a well-known Backpropagation Through Time (BPTT) method for calculation of dynamic derivatives is used. The idea of BPTT is unrolling the recurrent neural network back in time. In this way, RNN is presented as a deep feedforward network with shared weights where each layer is referring to one of retrospective time steps.

Then a standard backpropagation is applied to this deep network, immediate derivatives $\frac{\partial E}{\partial \mathbf{w}_{rec}(n)}$ are calculated. In derivations below we use a framework very similar to [20] but based on studying evolution of local gradients $\delta(k)$ for the backpropagation procedure.

The dynamic derivative is a sum of immediate derivatives: $\frac{\partial E}{\partial \mathbf{w}_{rec}} = \sum_{n=1}^{k} \frac{\partial E}{\partial \mathbf{w}_{rec}(n)}$, where $n = 1, \ldots, h$, where $h$ is BPTT’s truncation depth. An intermediate variable $\delta \equiv \frac{\partial E}{\partial \mathbf{w}_{rec}}$ called a “local gradients” or simply “deltas” is usually introduced for convenience. If deltas are known for specific layer $n$ then corresponding immediate derivatives can be obtained easily: $\frac{\partial E}{\partial \mathbf{w}_{rec}(n)} = (\mathbf{u}(n))^T \delta(n)$ and $\frac{\partial E}{\partial \mathbf{w}_{rec}(n)} = \mathbf{z}(n − 1)^T \delta(n)$.

For the last layer $\delta$ is an error residual, for the intermediate layers deltas are incrementally calculated according to very famous backpropagation formula:

$$
\delta_j(n − 1) = f'(a_j(n − 1)) \sum_i w_{rec,j}^i \delta_i(n).
$$

(2)

Let’s write this equation in a matrix form:

$$
\delta(n − 1) = \delta(n)^T \text{diag}(f'(\mathbf{a}(n − 1))),(3)
$$

where $\text{diag}$ converts a vector into diagonal matrix. Equation (3) may be rewritten using is Jacobian matrix $J(n) = \frac{\partial \mathbf{a}(n)}{\partial \mathbf{a}(n − 1)}$:

$$
\delta(n − 1) = \delta(n)J(n),
$$

(4)

where

$$
J(n) = \mathbf{w}_rec^T \text{diag}(f'(\mathbf{a}(n − 1))).
$$

(5)

Now we can use an intuitive understanding of exploding/vanishing gradients problem that was deeply investigated in classic [8], [23] and modern papers [18], [20]. As it can be seen from (5), norm of the backpropagated deltas is strongly dependent on norm of the Jacobians. Moreover, they actually are product of Jacobians: $\delta(n − h) = \delta(n)J(n(n − 1)) \cdots J(n − h + 1)$. The “older” deltas are, the more Jacobian matrices were multiplied. If norm of Jacobians are more than 1, the gradients will grow exponentially in most cases. It refers to the RNN’s behavior where long-term components are more important than short-term ones. Vice versa, if norm of Jacobians are less than 1, this leads to vanishing gradients and “forgetting” the long-term events.

At the same time, even if both factors in (5) have norm $1$, it still not guarantee the same norm of $J(n)$. In [20] the power iteration method was used to formally analyze product of Jacobian matrices and obtain tight conditions for when the
gradients explode or vanish. It was shown that it is sufficient for the largest eigenvalue of the recurrent weight matrix $w_{rec}$ to be smaller than 1 for long term components to vanish and necessary for it to be larger than 1 for gradients to explode.

III. GRADIENT REGULARIZATION

In [22] one can find an approach called “pseudoregularization” for forcing the recurrent neural networks to capture long-term behavior. The idea is to control the backpropagation flow during the training. The target cost function $L(w)$ was modified there in order to perform a multi-objective optimization by adding an additional term $\Omega(w)$ responsible for the size of backpropagated gradients:

$$L(w) = E(w) + \lambda \Omega(w), \quad (6)$$

where $E(w)$ is a target error function, $\Omega(w)$ is a regularizer that prevents “overvanishing” the gradients, $\lambda$ is a gradient regularization rate. This regularizer was a function of the deltas:

$$\Omega(w) = \sum_k ((\|\delta(w, k)\|) - 1)^2. \quad (7)$$

It’s goal was set to make a mean norm of deltas 1. For this purpose derivatives $\frac{\partial E}{\partial w_{rec}}$ were derived and used in gradients-based optimization algorithm. However, this approach shows unsatisfactory sensitivity from the gradients regularization rate $\lambda$ and has very capricious behavior of training in general.

Similar, but more advanced approach called “gradient regularization” [21] was independently proposed by [20]. They used the following regularizer:

$$\Omega(w) = \sum_k \left( \frac{\partial E}{\partial w_{rec}} \frac{\partial w_{rec}}{\partial k} \right)^2 \quad (8)$$

It forces the Jacobian matrices $\frac{\partial E}{\partial w_{rec}} \frac{\partial w_{rec}}{\partial k}$ to preserve norm in the relevant direction $\frac{\partial E}{\partial w_{rec}} \frac{\partial w_{rec}}{\partial k}$, not for any direction like in the works mentioned above. Also, these constraints are softer. However, analytic derivatives $\frac{\partial E}{\partial w_{rec}}$ for [6] are rather bulky; their calculation is a hard task. Authors used Theano [Bergstra, 2010] that has a built-in symbolic mathematical engine for automatic calculation of derivatives. However, such functionality is uncommon right now that complicates using this approach.

IV. DIFFERENTIATION OF THE GRADIENT’S NORM

Consider we have a mini-batch of training data $d = \{u_1; t_1; \ldots; u_N, t_N\}$ containing $N$ samples. We made a forward and backward passes using this mini-batch and calculated the derivative $dw = \frac{\partial E}{\partial w}$. Now we want to check whether applying $dw$ to the network’s weights $w$ minimizes or maximizes norm of the backpropagated gradients. We control $\|\delta_\cdot\|$ because it is clear that norms of the derivatives $\frac{\partial E}{\partial w_{rec}^i}$ and $\frac{\partial E}{\partial w_{rec}}$ are strongly dependent from norms of local gradients.

**Theorem 1:** Let $\partial w_{rec}$ is a recurrent weights matrix of Simple Recurrent Network. Assume we already did forward and backward passes and calculated the weights update $\partial w_{rec}$ for the recurrent matrix, $w_{rec}^{(i+1)} = w_{rec}^{(i)} + \partial w_{rec}$. Sufficient condition for increasing $\|\delta(k - h)\|_{F_{ro}}$ (norm of local gradients propagated h time steps back from the current time step k) is $dS > 0$, where

$$dS = (G, dG), \quad (9)$$

$$G = \left( \prod_{i=h}^1 diag(f'(a(k - i + 1)))w_{rec} \right) \delta(k), \quad (10)$$

$$dG = \sum_i^h \left( \left( \prod_{j=1}^i diag(f'(a(k - j + 1)))v \right) \delta(k) \right), \quad (11)$$

$\forall i \neq j$.

Similarly, sufficient condition for decreasing $\|\delta(k - h)\|_{F_{ro}}$ is $dS < 0$.

**Proof:** Consider function $S'(w_{rec})$ that is Frobenius (Euclidean) norm of local gradients [2]:

$$S'(w_{rec}) = \|\delta(k - h, w_{rec})\|. \quad (12)$$

Using [3, 4] we get:

$$\delta(k-h) = \delta(k)w_{rec}^Tdiag(f'(a(k - 1))) \ldots w_{rec}^Tdiag(f'(a(k - h + 1))). \quad (13)$$

For convenience, let’s denote $D_n \equiv diag(f'(a(n)))$ and change the indexes for time steps as follows: $(k-h)$-th step is the $I$-st time step, $(k-h+1)$-th is the $2$-nd time step, $k$-th is the $H$-th time step. So far, equation [11] becomes:

$$\delta(1) = \delta(H)w_{rec}^T D_{H-1}w_{rec}^T D_{H-2} \ldots w_{rec}^T D_1 \quad (14)$$

Since differentiation of the squared norm is simpler than differentiation of the norm itself, $\|A\|_{F_{ro}} = \|A^T\|_{F_{ro}}$, $(A \cdot B)^T = B^T A^T$, we may introduce an intermediate variable $G$:

$$G = D_1 w_{rec} D_2 w_{rec} \ldots D_H w_{rec} \delta(H). \quad (15)$$

So, we are interested in catching a behavior of function $S(w_{rec})$ around the current point, where

$$S(w_{rec}) = \|G\|^2_{F_{ro}}. \quad (16)$$

To understand behavior of $S$ let’s calculate a differential $dS$. The sign of $dS$ defines either $S$ increases or decreases; absolute value of $dS$ defines the speed. Thus, we obtain:

$$dS = 2(G, dG), \quad (17)$$

where $G$ is [13] and $dG$ is calculated as follows:

$$dG = \sum_{i=1}^H \delta(k) \cdot D_1 \cdot w_{rec} \ldots \cdot D_i \cdot dw_{rec} \ldots \cdot D_1 \cdot w_{rec}, \quad (18)$$

IEEE 3 | Page
or, in another form:

\[
dG = \sum_{i=1}^{H} \left( \prod_{j=1}^{H} D_{ij} \delta(k) \right),
\]

\[
v = dw_{rec}, \text{ if } i = j; \quad v = w_{rec}, \text{ if } i \neq j.
\]

(17)

The theorem is proven.

For calculation of (15) we use scalar product (,) that may be formally introduced as (A, B) = \sum_{i,j} a_{ij}b_{ij}. Another option is to reshape matrices to the vectors and use standard scalar products for vectors.

In this paper we propose a novel method for gradient regularization using the simple analytic criteria for selection of samples for training.

V. SAMPLING-BASED GRADIENT REGULARIZATION

It is generally known that training recurrent networks is a rather capricious process. Modification of cost function for preserving the norm of gradients causes difficulties even if gradients vectors are modified in relevant direction \(\frac{\partial E}{\partial \theta_{\text{par}}}\). Training of SRNs on sequences containing short-term dependencies using modified cost function \(\delta\) produces worse accuracy than training using default cost function. Clearly, difficulty of training RNNs is related to the complexity of error surface. Adding an additional goal of optimization may only to complicate the training process. Also, inaccurate gradients causes a natural for RNNs so-called “butterfly effect” where a little perturbations at the start leads to large divergences in the end of produced sequences. This is a reason why using popular for DNN regularization methods like dropout doesn’t work well for RNNs [24].

Idea of our method is selection of “proper” samples of data for training. As it was shown in previous section, now we can clearly understand an impact of each mini-batch of input data to the norm of backpropagated gradients. It is possible to propose different strategies of multi-objective optimization to prevent vanishing the gradients. However, in this paper we decided to use the simplest and the most straightforward method: we watch a norm of the gradients; if norm becomes very small, we omit such mini-batches of data, which decrease this norm. Vice versa, if norm becomes very large, we skip mini-batches increasing this norm even more. Note also that it is better to skip minibatches with large \(|dS|\): they can cause high “leaps” of the gradient norm and therefore its self-oscillations.

To evaluate quality of backpropagation let’s introduce a new auxiliary variable called Q-factor:

\[
Q(\delta, h) = \log_{10}(\frac{\|\delta(k)\|}{\|\delta(k-h)\|}),
\]

(18)

that measures how much the norm of the gradient is decreased during the backpropagation. For ideal caching of long-term dependencies Q-factor must be close to 0.

A. Computationally Efficient Cached Algorithm

Target differential \(dS\) which is responsible for the norm of backpropagated gradients includes two parts: \(G\) and \(dG\) (second and 3d formulae in [9] respectively). Let’s rewrite it in more simple form:

\[
dG = D_{1}w_{rec}...D_{H}w_{rec}\delta(H),
\]

\[
dG = \sum_{j=1}^{H} D_{j}dw_{rec}...D_{H}w_{rec}\delta(H),
\]

(19)

where \(H\) is equal to the horizon of BPTT.

Let’s estimate the complexity of algorithm if we calculate it in a straightforward way. We see that \(G\) has \(2H\) multiplications of matrices, so it’s complexity is \(2H \cdot O(N_{w}^{3})\) where size of recurrent matrix \(w_{rec}\) is \(N_{w} \cdot N_{w}\). However, \%50 of these matrices are diagonal matrices which have multiplication complexity \(O(N_{w}^{2})\). Therefore, total calculation complexity of \(G\) may be estimated as \(O(HN_{w}^{3})\). At the same time, \(dG\) consists of \(H\) summations of products of matrices. Their calculation complexity is \(O(HN_{w}^{3})\), so the total calculation complexity of \(dS\) is \(H(H+1) \cdot O(N_{w}^{3})\) or \(\approx O(H^{2}N_{w}^{3})\).

Meanwhile, we can propose much more numerically effective algorithm for calculation of \(dS\). It is clear from (19) that each summand of \(dG\) contains a lot of common parts which can be cached. Let’s introduce additional matrices \(L_{j}\) and \(R_{j}\) which are responsible for left and right parts against \(D_{j}dw_{rec}\) in (19):

\[
dG = \sum_{j=1}^{H} L_{j} (D_{j}dw_{rec}) R_{j},
\]

(20)

\(j = 1, ..., H\). Matrices \(L_{j}\) and \(R_{j}\) may be calculated recursively.

\[
L_{1} = I,
\]

\[
L_{j+1} = L_{j}D_{j}w_{rec},
\]

(21)

\[
R_{H} = I,
\]

\[
R_{j-1} = D_{j}w_{rec}R_{j},
\]

(22)

Thus, we need \(2(N_{w}^{3})\) operations for calculation \(L_{j}\) and \(R_{j}\) for each summand in (20). Also, we need \(2(N_{w}^{3})\) operations for multiplication of these matrices and \(D_{j}w_{rec}\) in (20). Finally, the complexity of algorithm now becomes \(O(HN_{w}^{3})\). This cached method is much more effective numerically in comparison to the straightforward method because it is linear function of horizon of backpropagation instead of quadratic function.

We performed a numerical experiment to test the performance of the methods discussed above. We used a neural network with 100 hidden units that is described in the “Experiments” section.

Algorithm description

We have a training data that contains train data \(\{U, T\}\). Let’s call an epoch a set of \(M\) weights updates. In our experiments one epoch means processing of 100 mini-batches.

IEEE 4 | Page
B. Description of experiments

For regression and softmax function. For output layer comparison, two sets containing 10 networks each were sampled from random numbers after the entire sequence was seen. The first marked. The model needs to predict the sum of the two numbers, where two random positions (one in the beginning and one in the middle of the sequence) are of random numbers, where two random positions (one in "Addition", "Multiplication", "Temporal order", "Temporal long-term correlations. We used four problems from this set:

A. Experimental tasks

We refer to [11] and [20] for description of experiment.

Algorithm 1 Main Algorithm

```
Input: training data \{U, T\}, r_0 > 0.
for each minibatch u_i; d_i, with N_{MD} vectors do
    calculate an the differential dS, if |dS| > 0
    break
    make forward and backward propagation for the network
    compute the Q factor Q(\delta, h) using formula [18]
    if Q(\delta, h) \in [Q_{min}; Q_{max}] then
        use current minibatch d_i to compute the derivative \frac{\partial E}{\partial w}
    else
        if (Q(\delta, h) < Q_{min} and dS > 0) or (Q(\delta, h) > Q_{max} and dS < 0) then
            use current minibatch for training
        else
            break
        end if
    end if
end for
```

VI. EXPERIMENTS

A. Experimental tasks

We refer to [11] and [20] for description of experiment. They used pathological synthetic test set from [2] that requires long-term correlations. We used four problems from this set: “Addition”, “Multiplication”, “Temporal order”, “Temporal order 3-bit”.

For “Addition problem” the input consists of a sequence of random numbers, where two random positions (one in the beginning and one in the middle of the sequence) are marked. The model needs to predict the sum of the two random numbers after the entire sequence was seen. The first position is sampled from [1; T’/10], while the second position is sampled from [T’/10; T’/2], where T’ is sampled from [T; 11/10T], T is the length of the sequence in the paper.

For description of other problems, please, see [20].

B. Description of experiments

We use SRNs with 100 hidden units and tanh activation function. For output layer linear activation function was used for regression and softmax for classification. To perform better comparison, two sets containing 10 networks each were initialized by random values and saved. Thus, for training using different methods we varied training algorithms only, initial weights of neural networks were the same. “Safe” range [Q_{MIN}; Q_{MAX}] was set to [-1; 1].

We use Stochastic Gradient Descent (SGD) as optimization algorithm, training speed \(\alpha = 10^{-5}...10^{-3}\), momentum \(\mu = 0.9\), size of mini-batch is 10. Dataset contains 20,000 samples for training, 1000 samples for validation and 10,000 samples for tests. Training process consists 2000 epochs, each epoch consists 50 iterations, i.e. 100,000 corrections of weights at all. Selection of the best copy of trained network is performed using “save best” method: after each epoch, network’s performance is tested on validation dataset; network that has the best performance on the validation dataset is tested on the test dataset, this result is recognized as the final result.

C. Experimental results

For recurrent neural networks good initialization of weights is very important because vanishing/exploding gradients has monotonous flow in most cases because gradients are propagated through the same matrix of recurrent weights. In our experiments weights were initialized by small values from Gaussian distribution with zero mean and standard deviation \(\sigma = 0.01\) as in [20]. On Fig. 4 average norms of gradients as function of backpropagation depth (before training, further referred as initial gradients) are graphed for different values \(\sigma\).

This experiment was made for the “Temporal order problem”.

Each chart at Fig. 4 contains three curves: average norms of local gradients \(\delta(k)\) (blue) and average norms of gradients \(\Delta w(k)_{in} = \frac{\partial E}{\partial w_{in}}\) and \(\Delta w(k)_{rec} = \frac{\partial E}{\partial w_{rec}}\) (red and green). From the graphs at Fig. 3.5 one can ensure on practice that to control the norms \(\frac{\partial E}{\partial w}\) which actually make changes to the weights and are under the main scope of our interest it is enough to control the norms of local gradients \(\delta(k)\) because they are highly correlated.

It can be seen from Fig. 4 that backpropagated gradient flow is very sensitive to the parameters of random initializing of weights. Size of standard deviation that was used for our experiments from [Pascanu, 2012] ensures smooth flow of gradients’ norms (Fig4, a), the Q-factor for this case is \(Q(\delta, h) \approx 0.7\) for \(h = 100\). This is good for catching both long-term and short-term dependencies. But decreasing or increasing of parameter \(\sigma\) causes catastrophic effects for initial gradients flow, (see Fig. 4b and c). For \(\sigma = 0.005\) average norm of gradients’ falls down during \(h = 100\) steps backwards to less than \(10^{-25}\), here \(Q(\delta, h) > 25\), it is classic.
case of vanishing gradients effect: “old” inputs don’t have impact on final error $E(w)$ and on training of neural network at all.

Vice versa, for $\sigma = 0.02$ average norm of gradients is explosively growing, here $Q(\delta, h) < -6$, only “old” values make important changes to network’s weights.

Poorly initialized networks have bad chances to be successfully trained on long-term temporal problems using 1-st order optimization methods because i) low sensitivity of 1-st order methods to small variations in gradients values and ii) local nature of gradient optimization methods at all.

However, proper initialization doesn’t guarantee successful training. Particular case of forward and backward dynamics during training of SRN network is shown on Fig. 5. Norms of the backpropagated gradients are depicted on the top, mean and median activation values are shown on the bottom of each subfigure there. Neural network that is depicted on Fig. 5 was initialized with $\sigma = 0.01$ and initial norms of backpropagated gradients were similar to Fig. 4 a). However, after 500 iterations we got norm of gradients less than $10^{-7}$ after $h=100$ backpropagations via BPTT. After that almost all the time neural networks had small gradients in the range $10^{-7}...10^{-8}$ . From the graphs on Fig. 5 on the left, we see that area of small gradients is related to area of saturation for neuron’s activations. This is a symptom of bad network abilities for successful training and obtaining good generalization properties.

Using gradient regularization enables keeping the norms of backpropagated gradients from vanishing or explosion. On Fig. 5 forward and backward dynamics of the previous neural network is shown; but now it is trained using our sampling-based gradient regularization. It follows from the graphs that norms of gradients are all time in the safe range, here $Q(\delta, h) \in [-1; 1]$. Graphs of activation values in (Fig. 5) shows more proper forward dynamics that is almost all the time are out of saturation area.

On Fig. 7 network’s performance for “Temporal order problem” with and without our proposed sampling-based gradient regularization for pre-initialized sets of 10 SRNs.

Using our sampling-based gradients regularization allows to refine quality of training. For lengths $T = 50$ and using finely selected hyperparameters we got accuracies up to almost 100% correct answers without using gradients regularization. For lengths $T = 100$ and $T = 150$ improvement is 10-20% in average. If we believe that model is successful if it produces more than 99% correct answers then for $T=100$ it was very crucial. Results for other experiments are shown in Table I.

For sequences of length $T = 100$ we were able to train at least one successful model (with accuracy >99% as required [Hochreiter, 1997]) for all four problems using the proposed method. For two problems (Adding and Temporal order) it was impossible using the traditional training. Unfortunately, we were not able to train the successful model for $T = 150$ neither using the proposed method nor using the traditional method. Meanwhile, sampling-based gradient regularization also improved best and mean accuracies almost for all cases.

Samples rejected by the algorithm during the training not necessarily are lost for using in future training process because
they may be used when network is in “safe region” or we may need to change norms of gradients in the opposite direction. Range for “safe” norms of gradients $Q(\delta, h) \in [-1; 1]$ was found empirically for our problems. For different problems and lengths of sequences it could be different.

VII. CONCLUSION

We provided a novel solution of the problem of exploding and vanishing gradient effects, applied to the Simple Recurrent Networks. Basing on estimation of the gradient norm’s differential we can predict each minibatch’s influence.

Using this technique we build the algorithm that controls magnitude of the gradient operating solely with presence of the minibatches in the training sequence. We have shown mathematical correctness of this algorithm, and introduced mechanisms of negative feedback that prevent self-oscillations inside the training process.

This frameworks was tested for long-term prediction on a comprehensive set of appropriate benchmarks. Resulting accuracy outperforms best known SRN learning algorithms by 10-20%. This paradigm could be generalized to deep and multi-layered recurrent networks, that is a subject of our future research.

ACKNOWLEDGMENTS

We thank FlyElephant (http://flyelephant.net) and Dmitry Spodares for computational resources kindly given for our experiments.

REFERENCES

[1] B.G. Horne H.T. Siegelmann. Computational capabilities of recurrent neural networks. IEEE Transactions on Systems, Man, and Cybernetics, B 27(2):208–215, 1997.

[2] H.G. Zimmermann A.M. Schfer. Recurrent neural networks are universal approximators. In Lecture Notes in Computer Science, International Conference on Artificial Neural Networks (ICANN), volume 4131, pages 632–640, 2006.

[3] H. Cardot R. Bone. Advanced Methods for Time Series Prediction Using Recurrent Neural Networks, chapter Advanced Methods for Time Series Prediction Using Recurrent Neural Networks, pages 1536. Intech. Croatia, 2011.

[4] D. V. Prokhorov. Toyota prius hnev neurocontrol and diagnostics. Neural Networks, 21:458–465, 2008.

[5] et. al T. Mikolov, M. Karafiat. Recurrent neural network based language model. In INterspeech, Makuhari, Chiba, Japan, 26-30 September 2010.

[6] G. Hinton A. Graves, A.R. Mohamed. Speech recognition with deep recurrent neural networks. In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Vancouver, Canada, 26-31 May 2013.

[7] D. Bahdanau K. Cho, B. van Merriënboer. On the properties of neural machine translation: Encoder-decoder approaches. In Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8), Doha, Qatar, October 2014.

[8] P. Frasconi Y. Bengio, P. Simard. Learning long-term dependencies with gradient descent is difficult. . IEEE Trans. Neural Networks, 5(2):157–166, 1994.

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

[10] S. Haykin, editor. Kalman Filtering and Neural Networks. John Wiley & Sons, 2001.

[11] I. Sutskever J. Martens. Learning recurrent neural networks with hessian-free optimization. In Proceedings of the 28th International Conference on Machine Learning (ICML), 2011.

[12] Herbert Jaeger. 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, 2001.

[13] H. Jaeger. Long short-term memory in echo state networks: Details of a simulation study. Technical Report 27, Jacobs University, 2012.

[14] EM Kussul and DA Rachkovskij. Multilevel assembly neural architecture and processing of sequences. Neurocomputers and Attention: Connectionism and neurocomputers, 2:577–590, 1991.

[15] L. Fei-Fei A. Karpathy. Deep visual-semantic alignments for generating image descriptions. In CVPR, 2015.

[16] I. Sutskever R. Jozefowicz, W. Zaremba. An empirical exploration of recurrent network architectures. In Proceedings of the 32-nd International Conference on Machine Learning (ICML), 2015.

[17] T. Mikolov. Statistical Language Models based on Neural Networks. Ph.d. thesis, Brno University of Technology, 2012.

[18] R. Pascanu Y. Bengio, N. Boulanger-Lewandowski. Advances in optimizing recurrent networks. In Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 8624–8628, Vancouver Canada, 26-31 May 2013.

[19] Ilya Sutskever. Training Recurrent Neural Networks. Ph.d. thesis, University of Toronto, 2013.

[20] Y. Bengio R. Pascanu. On the difficulty of training recurrent neural networks. Technical report, Universite de Montreal, 2012.

[21] A. Courville Y. Bengio, I. J. Goodfellow. Deep Learning. MIT Press, 2015. book in preparation.
[22] A.N. Chernodub. Training neuroemulators using multicriteria extended kalman filter and pseudoregularization for model reference adaptive neurocontrol. In Proceedings of IEEE IV International Congress on Ultra Modern Telecommunications and Control Systems (ICUMT), pages 397–402, St. Petersburg, Russia, 3-5 October 2012.

[23] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. Master’s thesis, TU Munich, 1991.

[24] O. Vinyals W. Zaremba, I. Sutskever. Recurrent neural network regularization. In ICLR 2015, 2015.
TABLE I: Accuracies of trained SRNs for synthetic problems which have long-term dependencies without gradient regularization (traditional training method) and with sampling-based gradient regularization (proposed method), T = 100.

|                      | Adding     | Multiplication | Temporal order | Temporal order 3-bit |
|----------------------|------------|----------------|----------------|-----------------------|
|                      | best | mean | best | mean | best | mean | best | mean |
| Without Gradient Regularization | 99% | 68% | >99% | 72% | 96% | 44% | 99% | 50% |
| Sampling-based Gradient Regularization | >99% | 96% | >99% | 68% | >99% | 60% | >99% | 62% |

TABLE II: Accuracies of trained SRNs for synthetic problems which have long-term dependencies without gradient regularization (traditional training method) and with sampling-based gradient regularization (proposed method), T = 150.

|                      | Adding     | Multiplication | Temporal order | Temporal order 3-bit |
|----------------------|------------|----------------|----------------|-----------------------|
|                      | best | mean | best | mean | best | mean | best | mean |
| Without Gradient Regularization | 34% | 11% | N/A | N/A | 51% | 30% | 32% | 24% |
| Sampling-based Gradient Regularization | 47% | 13% | N/A | N/A | 72% | 42% | 37% | 30% |