Recurrent Dropout without Memory Loss

Stanislau Semeniuta[1] **Aliaksei Severyn**[2] **Erhardt Barth**[1]
[1]Universität zu Lübeck, Institut für Neuro- und Bioinformatik
{stas,barth}@inb.uni-luebeck.de
[2]Google Research
severyn@google.com

Abstract

This paper presents a novel approach to recurrent neural network (RNN) regularization. Differently from the widely adopted dropout method, which is applied to *forward* connections of feed-forward architectures or RNNs, we propose to drop neurons directly in *recurrent* connections in a way that does not cause loss of long-term memory. Our approach is as easy to implement and apply as the regular feed-forward dropout and we demonstrate its effectiveness for Long Short-Term Memory network, the most popular type of RNN cells . Our experiments on three NLP benchmarks show consistent improvements even when combined with conventional feed-forward dropout.

1 Introduction

Recurrent Neural Networks, LSTMs in particular, have recently become a popular tool among NLP researchers for their superior ability to model and learn from sequential data. These models have shown state-of-the-art results on various public benchmarks ranging from sentence classification (Wang et al., 2015; Irsoy and Cardie, 2014; Liu et al., 2015) and various tagging problems (Dyer et al., 2015) to language modelling (Kim et al., 2015; Zhang et al., 2015), text generation (Zhang and Lapata, 2014) and sequence-to-sequence prediction tasks (Sutskever et al., 2014).

Having shown excellent ability to capture and learn complex linguistic phenomena, RNN architectures are prone to overfitting. Among the most widely used techniques to avoid overfitting in neural networks is the dropout regularization (Hinton et al., 2012). Since its introduction it has become, together with the L2 weight decay, the standard method for neural network regularization. While showing significant improvements when used in feed-forward architectures, e.g., Convolutional Neural Networks (Krizhevsky et al., 2012), the application of dropout in RNNs has been somewhat limited. Indeed, so far dropout in RNNs has been applied in the same fashion as in feed-forward architectures: it is typically injected in input-to-hidden and hidden-to-output connections, i.e., along the input axis, but not between the recurrent connections (time axis). Given that RNNs are mainly used to model sequential data with the goal of capturing short- and long-term interactions, it seems natural to also regularize the recurrent weights. This observation has led us and other researchers (Moon et al., 2015; Gal, 2015) to the idea of applying dropout to the recurrent connections in RNNs.

In this paper we propose a novel *recurrent dropout* technique and demonstrate how our method is superiour to other recurrent dropout methods recently proposed in (Moon et al., 2015; Gal, 2015). Additionally, we answer the following questions which helps to understand how to best apply recurrent dropout: (i) how to apply the dropout in recurrent connections of the LSTM architecture in a way that prevents possible corruption of the long-term memory; (ii) what is the relationship between our *recurrent dropout* and the widely adopted dropout in input-to-hidden and hidden-to-output connections; (iii) how the dropout mask in RNNs should be sampled: once per step or once per sequence. The latter question of sampling the mask appears to be crucial in some cases to make the recurrent dropout work and, to

Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 1757–1766, Osaka, Japan, December 11-17 2016.

the best of our knowledge, has received very little attention in the literature. Our work is the first one to provide empirical evaluation of the differences between these two sampling approaches.

Regarding empirical evaluation, we first highlight the problem of information loss in memory cells of LSTMs when applying *recurrent dropout*. We demonstrate that previous approaches of dropping *hidden state* vectors cause loss of memory while our proposed method to use dropout mask in *hidden state update* vectors does not suffer from this problem. We experiment on three widely adopted NLP tasks: word- and character-level Language Modeling and Named Entity Recognition. The results demonstrate that our *recurrent dropout* helps to achieve better regularization and yields improvements across all the tasks, even when combined with the conventional feed-forward dropout. Furthermore, we compare our dropout scheme with the recently proposed alternative recurrent dropout methods and show that our technique is superior in almost all cases.

2 Related Work

Neural Network models often suffer from overfitting, especially when the number of network parameters is large and the amount of training data is small. This has led to a lot of research directed towards improving their generalization ability. Below we primarily discuss some of the methods aimed at improving regularization of RNNs.

Pham et al. (2013) and Zaremba et al. (2014) have shown that LSTMs can be effectively regularized by using dropout in forward connections. While this already allows for effective regularization of recurrent networks, it is intuitive that introducing dropout also in the hidden state may force it to create more robust representations. Indeed, Moon et al. (2015) have extended the idea of dropping neurons in forward direction and proposed to drop cell states as well showing good results on a Speech Recognition task. Bluche et al. (2015) carry out a study to find where dropout is most effective, e.g. input-to-hidden or hidden-to-output connections. The authors conclude that it is more beneficial to use it once in the correct spot, rather than to put it everywhere. Bengio et al. (2015) have proposed an algorithm called scheduled sampling to improve performance of recurrent networks on sequence-to-sequence labeling tasks. A disadvantage of this work is that the scheduled sampling is specifically tailored to this kind of tasks, what makes it impossible to use in, for example, sequence-to-label tasks. Gal (2015) uses insights from variational Bayesian inference to propose a variant of LSTM with dropout that achieves consistent improvements over a baseline architecture without dropout.

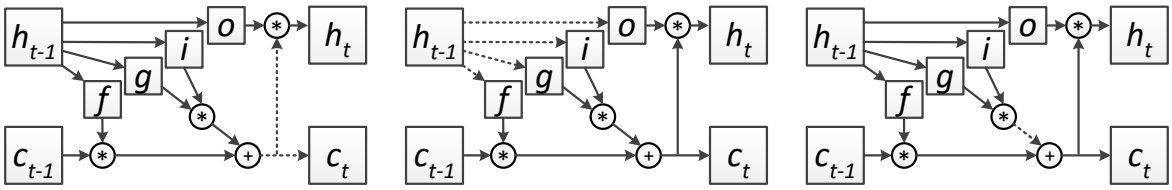
The main contribution of this paper is a new *recurrent dropout* technique, which is most useful in gated recurrent architectures such as LSTMs and GRUs. We demonstrate that applying dropout to arbitrary vectors in LSTM cells may lead to loss of memory thus hindering the ability of the network to encode long-term information. In other words, our technique allows for adding a strong regularizer on the model weights responsible for learning short and long-term dependencies without affecting the ability to capture long-term relationships, which are especially important to model when dealing with natural language. Finally, we compare our method with alternative *recurrent dropout* methods recently introduced in (Moon et al., 2015; Gal, 2015) and demonstrate that our method allows to achieve better results.

3 Recurrent Dropout

In this section we first show how the idea of feed-forward dropout (Hinton et al., 2012) can be applied to recurrent connections in vanilla RNNs. We then introduce our *recurrent dropout* method specifically tailored for gated architectures such as LSTMs and GRUs. We draw parallels and contrast our approach with alternative recurrent dropout techniques recently proposed in (Moon et al., 2015; Gal, 2015) showing that our method is favourable when considering potential memory loss issues in long short-term architectures.

3.1 Dropout in vanilla RNNs

Vanilla RNNs process the input sequences as follows:



(a) Moon et al., 2015 (b) Gal, 2015 (c) Ours

Figure 1: Illustration of the three types of dropout in recurrent connections of LSTM networks. Dashed arrows refer to dropped connections. Input connections are omitted for clarity.

$$\mathbf{h_t} = f(\mathbf{W_h}[\mathbf{x_t}, \mathbf{h_{t-1}}] + \mathbf{b_h}), \tag{1}$$

where $\mathbf{x_t}$ is the input at time step t; $\mathbf{h_t}$ and $\mathbf{h_{t-1}}$ are hidden vectors that encode the current and previous states of the network; $\mathbf{W_h}$ is parameter matrix that models input-to-hidden and hidden-to-hidden (recurrent) connections; $\mathbf{b}$ is a vector of bias terms, and f is the activation function.

As RNNs model sequential data by a fully-connected layer, dropout can be applied by simply dropping the previous hidden state of a network. Specifically, we modify Equation 1 in the following way:

$$\mathbf{h_t} = f(\mathbf{W_h}[\mathbf{x_t}, d(\mathbf{h_{t-1}})] + \mathbf{b_h}), \tag{2}$$

where d is the dropout function defined as follows:

$$d(\mathbf{x}) = \begin{cases} mask * \mathbf{x}, & \text{if train phase} \\ (1-p)\mathbf{x} & \text{otherwise,} \end{cases} \tag{3}$$

where p is the dropout rate and $mask$ is a vector, sampled from the Bernoulli distribution with success probability $1 - p$.

3.2 Dropout in LSTM networks

Long Short-Term Memory networks (Hochreiter and Schmidhuber, 1997) have introduced the concept of gated inputs in RNNs, which effectively allow the network to preserve its memory over a larger number of time steps during both forward and backward passes, thus alleviating the problem of vanishing gradients (Bengio et al., 1994). Formally, it is expressed with the following equations:

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma(\mathbf{W}_i \left[\mathbf{x}_t, \mathbf{h}_{t-1}\right] + \mathbf{b}_i) \\ \sigma(\mathbf{W}_f \left[\mathbf{x}_t, \mathbf{h}_{t-1}\right] + \mathbf{b}_f) \\ \sigma(\mathbf{W}_o \left[\mathbf{x}_t, \mathbf{h}_{t-1}\right] + \mathbf{b}_o) \\ f(\mathbf{W}_g \left[\mathbf{x}_t, \mathbf{h}_{t-1}\right] + \mathbf{b}_g) \end{pmatrix} \tag{4}$$

$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \mathbf{g}_t \tag{5}$$

$$\mathbf{h}_t = \mathbf{o}_t * f(\mathbf{c}_t), \tag{6}$$

where $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$ are input, output and forget gates at step t; $\mathbf{g}_t$ is the vector of cell updates and $\mathbf{c}_t$ is the updated cell vector used to update the hidden state $\mathbf{h}_t$; σ is the sigmoid function and $*$ is the element-wise multiplication.

Gal (2015) proposes to drop the previous *hidden state* vectors when computing values of gates and updates of the current step, where he samples the dropout mask once for every sequence:

$$\begin{pmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} \sigma(\mathbf{W}_i \left[\mathbf{x}_t, d(\mathbf{h}_{t-1})\right] + \mathbf{b}_i) \\ \sigma(\mathbf{W}_f \left[\mathbf{x}_t, d(\mathbf{h}_{t-1})\right] + \mathbf{b}_f) \\ \sigma(\mathbf{W}_o \left[\mathbf{x}_t, d(\mathbf{h}_{t-1})\right] + \mathbf{b}_o) \\ f(\mathbf{W}_g \left[\mathbf{x}_t, d(\mathbf{h}_{t-1})\right] + \mathbf{b}_g) \end{pmatrix} \tag{7}$$

Moon et al. (2015) propose to apply dropout directly to the cell values and use per-sequence sampling as well:

$$\mathbf{c}_t = d(\mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \mathbf{g}_t) \tag{8}$$

In contrast to dropout techniques proposed by Gal (2015) and Moon et al. (2015), we propose to apply dropout to the *cell update* vector $\mathbf{g}_t$ as follows:

$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * d(\mathbf{g}_t) \tag{9}$$

Different from methods of (Moon et al., 2015; Gal, 2015), our approach does not require sampling of the dropout masks once for every training sequence. On the contrary, as we will show in Section 4, networks trained with a dropout mask sampled per-step achieve results that are at least as good and often better than per-sequence sampling. Figure 1 shows differences between approaches to dropout.

The approach of (Gal, 2015) differs from ours in the overall strategy – they consider network's hidden state as input to subnetworks that compute gate values and cell updates and the purpose of dropout is to regularize these subnetworks. Our approach considers the architecture as a whole with the hidden state as its key part and regularize the whole network. The approach of (Moon et al., 2015) on the other hand is seemingly similar to ours. In Section 3.3 we argue that our method is a more principled way to drop recurrent connections in gated architectures.

It should be noted that while being different, the three discussed dropout schemes are not mutually exclusive. It is in general possible to combine our approach and the other two. We expect the merge of our scheme and that of (Gal, 2015) to hold the biggest potential. The relations between recurrent dropout schemes are however out of scope of this paper and we rather focus on studying the relationships of different dropout approaches with the conventional forward dropout.

Lastly, we note that our dropout is also applicable to the recently introduced Gated Recurrent Unit (GRU) networks (Cho et al., 2014). GRU networks are built on the same design principles as LSTM networks and our dropout technique applies in a similar fashion.

3.3 Dropout and memory

We found that an intuitive idea to drop previous hidden states directly, as proposed in Moon et al. (2015), produces mixed results. We have observed that it helps the network to generalize better when not coupled with the forward dropout, but is usually no longer beneficial when used together with a regular forward dropout.

The problem is caused by the scaling of neuron activations during inference. Consider the hidden state update rule in the test phase of an LSTM network. For clarity, we assume every gate to be equal to 1:

$$\mathbf{h}_t = (\mathbf{h}_{t-1} + \mathbf{g}_t)p, \tag{10}$$

where $\mathbf{g}_t$ are update vectors computed by Eq. 4 and p is the probability to not drop a neuron. As h_{t-1} was, in turn, computed using the same rule, we can rewrite this equation as:

$$\mathbf{h}_t = ((\mathbf{h}_{t-2} + \mathbf{g}_{t-1})p + \mathbf{g}_t)p \tag{11}$$

Recursively expanding $\mathbf{h}$ for every timestep results in the following equation:

$$\mathbf{h}_t = ((((\mathbf{h}_0 + \mathbf{g}_0)p + \mathbf{g}_1)p + ...)p + \mathbf{g}_t)p \tag{12}$$

Pushing p inside parenthesis, Eq. 12 can be written as:

$$\mathbf{h}_t = p^{t+1}\mathbf{h}_0 + \sum_{i=0}^{t} p^{t-i+1}\mathbf{g}_i \tag{13}$$

Since p is a value between zero and one, sum components that are far away in the past are multiplied by a very low value and are effectively removed from the summation. Thus, even though the network is

able to learn long-term dependencies, it is not capable of exploiting them during test phase. Note that our assumption of all gates being equal to 1 helps the network to preserve hidden state, since in a real network gate values lie within (0, 1) interval. In practice trained networks tend to saturate gate values (Karpathy et al., 2015) what makes gates to behave as binary switches. The fact that Moon et al. (2015) have achieved an improvement can be explained by the experimentation domain. Le et al. (2015) have proposed a simple yet effective way to initialize vanilla RNNs and reported that they have achieved a good result in the Speech Recognition domain while having an effect similar to the one caused by Eq. 13. One can reduce the influence of this effect by selecting a low dropout rate. This solution however is partial, since it only increases the number of steps required to completely forget past history and does not remove the problem completely.

One important note is that the dropout function from Eq. 3 can be implemented as:

$$d(\mathbf{x}) = \begin{cases} mask * \mathbf{x}/p, & \text{if train phase} \\ \mathbf{x} & \text{otherwise} \end{cases} \tag{14}$$

In this case the above argument holds as well, but instead of observing exponentially decreasing hidden states during testing, we will observe exponentially increasing values of hidden states during training.

Our approach addresses the problem discussed previously by dropping the update vectors $\mathbf{g}$. Since we drop only candidates, we do not scale the hidden state directly. This allows for solving the scaling issue, as Eq. 13 becomes:

$$\mathbf{h}_t = p\mathbf{h}_0 + \sum_{i=0}^{t} p\ \mathbf{g}_i = p\mathbf{h}_0 + p\sum_{i=0}^{t} \mathbf{g}_i \tag{15}$$

Moreover, since we only drop differences that are added to the network's hidden state at each time-step, this dropout scheme allows us to use per-step mask sampling while still being able to learn long-term dependencies. Thus, our approach allows to freely apply dropout in the recurrent connections of a gated network without hindering its ability to process long-term relationships.

We note that the discussed problem does not affect vanilla RNNs because they overwrite their hidden state at every timestep. Lastly, the approach of Gal (2015) is not affected by the issue as well.

4 Experiments

First, we empirically demonstrate the issues linked to memory loss when using various dropout techniques in recurrent nets (see Sec. 3.3). For this purpose we experiment with training LSTM networks on one of the synthetic tasks from (Hochreiter and Schmidhuber, 1997), specifically the Temporal Order task. We then validate the effectiveness of our *recurrent dropout* on three public benchmarks: word and character-level Language Modeling and Named Entity Recognition comparing directly to alternative *recurrent dropout* methods from (Moon et al., 2015; Gal, 2015).

4.1 Synthetic Task

Data. In this task the input sequences are generated as follows: all but two elements in a sequence are drawn randomly from {C, D} and the remaining two symbols from {A, B}. Symbols from {A, B} can appear at any position in the sequence. The task is to classify a sequence into one of four classes ({AA, AB, BA, BB}) based on the order of the symbols. We generate data so that every sequence is split into three parts with the same size and emit one meaningful symbol in first and second parts of a sequence. The prediction is taken after the full sequence has been processed. We use two modes in our experiments: `Short` with sequences of length 15 and `Medium` with sequences of length 30.

Setup. We use LSTM with one layer that contains 256 hidden units and *recurrent dropout* with 0.5 strength. Network is trained by SGD with a learning rate of 0.1 for 5k epochs. The networks are trained on 200 mini-batches with 32 sequences and tested on 10k sequences.

Results. Table 1 reports the results on the Temporal Order task when *recurrent dropout* is applied using our method and methods from (Moon et al., 2015) and (Gal, 2015). Using dropout from (Moon et al., 2015) with per-sequence sampling, networks are able to discover the long-term dependency, but fail to

Sampling	Moon et al. (2015)				Gal (2015); Ours			
	short sequences		medium sequences		short sequences		medium sequences	
	Train	Test	Train	Test	Train	Test	Train	Test
per-step	100%	100%	25%	25%	100%	100%	100%	100%
per-sequence	100%	25%	100%	<25%	100%	100%	100%	100%

Table 1: Accuracies on the Temporal Order task.

Dropout rate	Sampling	Moon et al. (2015)		Gal (2015)		Ours	
		Valid	Test	Valid	Test	Valid	Test
0.0	–	130.0	125.2	130.0	125.2	130.0	125.2
0.25	per-step	**113.0**	**108.7**	119.8	114.2	106.1	100.0
0.5	per-step	124.0	116.5	**118.3**	**112.5**	102.8	98.0
0.25	per-sequence	121.0	113.0	120.5	114.0	106.3	100.7
0.5	per-sequence	137.7	126.2	125.2	117.9	**103.2**	**96.8**
0.0	–	**94.1**	**89.5**	94.1	89.5	94.1	89.5
0.25	per-step	113.5	105.8	**92.9**	**88.4**	**91.6**	**87.0**
0.5	per-step	140.6	130.1	98.6	92.5	100.6	95.5
0.25	per-sequence	105.7	99.9	94.5	89.7	92.4	87.6
0.5	per-sequence	125.4	117.4	98.4	92.5	107.8	101.8

Table 2: Perplexity scores of the LSTM network on word level Language Modeling task (lower is better). Upper and lower parts of the table report results without and with forward dropout respectively. Networks with forward dropout use 0.2 and 0.5 dropout rates in input and output connections respectively. Values in bold show best results for each of the recurrent dropout schemes with and without forward dropout.

use it on the test set due to the scaling issue. Interestingly, in `Medium` case results on the test set are worse than random. Networks trained with per-step sampling exhibit different behaviour: in `Short` case they are capable of capturing the temporal dependency and generalizing to the test set, but require 10-20 times more iterations to do so. In `Medium` case these networks do not fit into the allocated number of iterations. This suggests that applying dropout to hidden states as suggested in (Moon et al., 2015) corrupts memory cells hindering the long-term memory capacity of LSTMs.

In contrast, using our *recurrent dropout* methods, networks are able to solve the problem in all cases. We have also ran the same experiments for longer sequences, but found that the results are equivalent to the `Medium` case. We also note that the approach of (Gal, 2015) does not seem to exhibit the memory loss problem.

4.2 Word Level Language Modeling

Data. Following Mikolov et al. (2011) we use the Penn Treebank Corpus to train our Language Modeling (LM) models. The dataset contains approximately 1 million words and comes with pre-defined training, validation and test splits, and a vocabulary of 10k words.

Setup. In our LM experiments we use recurrent networks with a single layer with 256 cells. Network parameters were initialized uniformly in [-0.05, 0.05]. For training, we use plain SGD with batch size 32 with the maximum norm gradient clipping (Pascanu et al., 2013). Learning rate, clipping threshold and number of Backpropagation Through Time (BPTT) steps were set to 1, 10 and 35 respectively. For the learning rate decay we use the following strategy: if the validation error does not decrease after each epoch, we divide the learning rate by 1.5. The aforementioned choices were largely guided by the work of Mikolov et al. (2014). To ease reproducibility of our results on the LM and synthetic tasks, we have

Dropout rate	Sampling	Moon et al. (2015)		Gal (2015)		Ours	
		Valid	Test	Valid	Test	Valid	Test
0.0	–	1.460	1.457	1.460	1.457	1.460	1.457
0.25	per-step	1.435	1.394	1.345	1.308	**1.338**	**1.301**
0.5	per-step	1.610	1.561	1.387	1.348	1.355	1.316
0.25	per-sequence	**1.433**	**1.390**	**1.341**	**1.304**	1.356	1.319
0.5	per-sequence	1.691	1.647	1.408	1.369	1.496	1.450
0.0	–	**1.362**	**1.326**	**1.362**	**1.326**	1.362	1.326
0.25	per-step	1.471	1.428	1.381	1.344	**1.358**	**1.321**
0.5	per-step	1.668	1.622	1.463	1.425	1.422	1.380
0.25	per-sequence	1.455	1.413	1.387	1.348	1.403	1.363
0.5	per-sequence	1.681	1.637	1.477	1.435	1.567	1.522

Table 3: Bit-per-character scores of the LSTM network on character level Language Modelling task (lower is better). Upper and lower parts of the table report results without and with forward dropout respectively. Networks with forward dropout use 0.2 and 0.5 dropout rates in input and output connections respectively. Values in bold show best results for each of the recurrent dropout schemes with and without forward dropout.

released the source code of our experiments[1].

Results. Table 2 reports the results for LSTM networks. We also present results when the dropout is applied directly to hidden states as in (Moon et al., 2015) and results of networks trained with the dropout scheme of (Gal, 2015). In addition, we report results of networks trained with no regularization and with dropout in only forward connetions in first rows of upper and lower parts of the table respectivelly. We make the following observations: (i) our approach shows better results than the alternatives; (ii) per-step mask sampling is better when dropping hidden state directly; (iii) on this task our method using per-step sampling seems to yield results similar to per-sequence sampling; (iv) in this case forward dropout yields better results than any of the three recurrent dropouts; and finally (v) both our approach and that of (Gal, 2015) are effective when combined with the forward dropout, though ours is more effective.

4.3 Character Level Language Modeling

Data. We train our networks on the dataset described in the previous section. It contains approximately 6 million characters, and a vocabulary of 50 characters. We use the provided partitions train, validation and test partitions.

Setup. We use networks with 1024 units to solve the character level LM task. The characters are embedded into 256 dimensional space before being processed by the LSTM. All parameters of the networks are initialized uniformly in [-0.01, 0.01]. We train our networks on non-overlapping sequences of 100 characters. The networks are trained with the Adam (Kingma and Ba, 2014) algorithm with initial learning rate of 0.001 for 50 epochs. We decrease the learning rate by 0.97 after every epoch starting from epoch 10. To avoid exploding gradients, we use MaxNorm gradient clipping with threshold set to 10.

Results. Results of our experiments are given in Table 3. Note that on this task regularizing only the recurrent connections is more beneficial than only the forward ones. In particular, LSTM networks trained with our approach and the approach of (Gal, 2015) yield a lower bit-per-character (bpc) score than those trained with forward dropout onlyWe attribute it to pronounced long term dependencies. In addition, our approach is the only one that improves over baseline LSTM with forward dropout. The overall best result is achieved by a network trained with our dropout with 0.25 dropout rate and per-step sampling, closely followed by network with Gal (2015) dropout.

[1] https://github.com/stas-semeniuta/drop-rnn

Dropout rate	Sampling	Moon et al. (2015)		Gal (2015)		Ours	
		Valid	Test	Valid	Test	Valid	Test
0.0	–	88.56	84.46	88.56	84.46	88.56	84.46
0.25	per-step	**88.79**	**84.80**	88.95	84.34	89.27	84.78
0.5	per-step	88.68	84.43	88.66	84.33	89.06	84.39
0.25	per-sequence	88.71	84.33	**88.54**	**84.88**	**89.32**	**84.95**
0.5	per-sequence	88.06	83.92	89.05	84.22	88.94	84.32
0.0	–	90.53	86.99	90.53	86.99	90.53	86.99
0.25	per-step	90.86	87.19	91.06	87.05	91.02	87.03
0.5	per-step	90.71	87.03	**90.76**	**87.23**	90.78	87.31
0.25	per-sequence	**90.73**	**87.32**	90.86	86.89	**90.99**	**87.33**
0.5	per-sequence	89.61	86.39	90.76	86.68	90.40	86.82

Table 4: F1 scores (higher is better) of the LSTM network on NER task (average scores over 3 runs). Upper and lower parts of the table report results without and with forward dropout respectively. Values in bold show best results for each of the recurrent dropout schemes with and without forward dropout.

4.4 Named Entity Recognition

Data. To assess our recurrent Named Entity Recognition (NER) taggers when using *recurrent dropout* we use a public benchmark from CONLL 2003 (Tjong Kim Sang and De Meulder, 2003). The dataset contains approximately 300k words split into train, validation and test partitions. Each word is labeled with either a named entity class it belongs to, such as `Location` or `Person`, or as being not named. The majority of words are labeled as not named entities. The vocabulary size is about 22k words.

Setup. Previous state-of-the-art NER systems have shown the importance of using word context features around entities. Hence, we slightly modify the architecture of our recurrent networks to consume the context around the target word by preprocessing the inputs by a convolutional layer. The size of the convolutional kernel is fixed to 5 words (the word to be labeled, two words before and two words after) and the number of filters is fixed to 256. The recurrent layer size is 1024 units. The network inputs include word embeddings (initialized with pretrained word2vec embeddings (Mikolov et al., 2013) and kept static) and capitalization features. For training we use the Adam algorithm (Kingma and Ba, 2014) with initial learning rate of 0.001. We train for 50 epochs and multiply the learning rate by 0.95 after every epoch starting at epoch 10. We also combine our *recurrent dropout* with the conventional forward dropout with the rate 0.2 in input and 0.5 in output connections. Lastly, we found that using $relu(x) = max(x, 0)$ nonlinearity resulted in higher performance than $tanh(x)$. We train our network on randomly extracted samples up to 15 words long and use full sentences for testing.

Results. Table 4 reports the results of networks trained with and without forward dropout and compares our algorithm to approaches of (Moon et al., 2015) and (Gal, 2015). We make the following observations: (i) forward dropout provides a much bigger improvement than recurrent one, what can be explained by the fact that long term dependencies are much less important in the NER task, in contrast to the Language Modeling; (ii) the results of our approach and dropout of (Gal, 2015) are comparable and both better than those of (Moon et al., 2015); and (iii) all three approaches consistently outperform baseline networks without dropout in recurrent connections.

5 Conclusions

This paper presents a novel *recurrent dropout* method specifically tailored to the gated recurrent neural networks. Our approach is easy to implement and is even more effective when combined with conventional forward dropout. We have shown that applying dropout to arbitrary cell vectors results in suboptimal performance. We discuss in detail the cause of this effect and propose a simple solution to overcome it. The effectiveness of our approach is verified on three public NLP benchmarks.

Our findings along with our empirical results help us to answer the questions posed in Section 1: (i) while is straight-forward to use dropout in vanilla RNNs due to their strong similarity with the feed-forward architectures, its application to LSTM networks is not so straightforward. We demonstrate that *recurrent dropout* is most effective when applied to *hidden state update* vectors in LSTMs rather than to *hidden states*; (ii) we observe an improvement in the network's performance when our *recurrent dropout* is coupled with the standard forward dropout, though the extent of this improvement depends on the values of dropout rates; (iii) per-step mask sampling is at least as good as per-sequence mask sampling when using our *recurrent dropout* method, with the most pronounced difference in the character level LM experiments, while the results of (Moon et al., 2015) and (Gal, 2015) are mixed.

Acknowledgments

This project has received funding from the European Union's Framework Programme for Research and Innovation HORIZON 2020 (2014-2020) under the Marie Skodowska-Curie Agreement No. 641805. Stanislau Semeniuta thanks the support from Pattern Recognition Company GmbH. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research.

References

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. *CoRR*, abs/1506.03099.

Theodore Bluche, Christopher Kermorvant, and Jérôme Louradour. 2015. Where to apply dropout in recurrent neural networks for handwriting recognition? In *13th International Conference on Document Analysis and Recognition, ICDAR 2015, Tunis, Tunisia, August 23-26, 2015*, pages 681–685.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and A. Noah Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *ACL*, pages 334–343. Association for Computational Linguistics.

Yarin Gal. 2015. A theoretically grounded application of dropout in recurrent neural networks. *arXiv:1512.05287*.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

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

Ozan Irsoy and Claire Cardie. 2014. Opinion mining with deep recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 720–728. Association for Computational Linguistics.

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. 2015. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2015. Character-aware neural language models. *CoRR*, abs/1508.06615.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. 2015. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941.

Pengfei Liu, Xipeng Qiu, Xinchi Chen, Shiyu Wu, and Xuanjing Huang. 2015. Multi-timescale long short-term memory neural network for modelling sentences and documents. In *ACL*. Association for Computational Linguistics.

T. Mikolov, S. Kombrink, L. Burget, J.H. Cernocky, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531, May.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michaël Mathieu, and Marc'Aurelio Ranzato. 2014. Learning longer memory in recurrent neural networks. *CoRR*, abs/1412.7753.

Taesup Moon, Heeyoul Choi, Hoshik Lee, and Inchul Song. 2015. Rnndrop: A novel dropout for rnns in asr. *Automatic Speech Recognition and Understanding (ASRU)*.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1310–1318.

Vu Pham, Christopher Kermorvant, and Jérôme Louradour. 2013. Dropout improves recurrent neural networks for handwriting recognition. *CoRR*, abs/1312.4569.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, pages 142–147, Stroudsburg, PA, USA. Association for Computational Linguistics.

Xin Wang, Yuanchao Liu, Chengjie SUN, Baoxun Wang, and Xiaolong Wang. 2015. Predicting polarities of tweets by composing word embeddings with long short-term memory. In *ACL*, pages 1343–1353. Association for Computational Linguistics.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *CoRR*, abs/1409.2329.

Xingxing Zhang and Mirella Lapata. 2014. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680. Association for Computational Linguistics.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2015. Tree recurrent neural networks with application to language modeling. *CoRR*, abs/1511.00060.