

Predefined Sparseness in Recurrent Sequence Models

Thomas Demeester, Johannes Deleu, Frédéric Godin, Chris Develder

Ghent University - imec

Ghent, Belgium

firstname.lastname@ugent.be

Abstract

Inducing sparseness while training neural networks has been shown to yield models with a lower memory footprint but similar effectiveness to dense models. However, sparseness is typically induced starting from a dense model, and thus this advantage does not hold during training. We propose techniques to enforce sparseness upfront in recurrent sequence models for NLP applications, to also benefit training. First, in language modeling, we show how to increase hidden state sizes in recurrent layers without increasing the number of parameters, leading to more expressive models. Second, for sequence labeling, we show that word embeddings with predefined sparseness lead to similar performance as dense embeddings, at a fraction of the number of trainable parameters.

1 Introduction

Many supervised learning problems today are solved with deep neural networks exploiting large-scale labeled data. The computational and memory demands associated with the large amount of parameters of deep models can be alleviated by using *sparse* models. Applying sparseness can be seen as a form of regularization, as it leads to a reduced amount of model parameters¹, for given layer widths or representation sizes. Current successful approaches gradually induce sparseness during training, starting from densely initialized networks, as detailed in Section . However, we propose that models can also be built with *predefined sparseness*, i.e., such models are already sparse by design and do not require sparseness inducing training schemes.

¹The sparseness focused on in this work, occurs on the level of trainable parameters, i.e., we do not consider data sparsity.

The main benefit of such an approach is memory efficiency, even at the start of training. Especially in the area of natural language processing, in line with the hypothesis by Yang et al. (2017) that natural language is “high-rank”, it may be useful to train larger sparse representations, even when facing memory restrictions. For example, in order to train word representations for a large vocabulary using limited computational resources, predefined sparseness would allow training larger embeddings more effectively compared to strategies inducing sparseness from dense models.

The contributions of this paper are (i) a predefined sparseness model for recurrent neural networks, (ii) as well as for word embeddings, and (iii) proof-of-concept experiments on part-of-speech tagging and language modeling, including an analysis of the memorization capacity of dense vs. sparse networks. An overview of related work is given in the next Section . We subsequently present predefined sparseness in recurrent layers (Section), as well as embedding layers (Section), each illustrated by experimental results. This is followed by an empirical investigation of the memorization capacity of language models with predefined sparseness (Section). Section summarizes the results, and points out potential areas of follow-up research.

The code for running the presented experiments is publicly available.²

2 Related Work

A substantial body of work has explored the benefits of using sparse neural networks. In deep convolutional networks, common approaches include sparseness regularization, e.g., using decomposition (Liu et al., 2015) or variational dropout (Molchanov et al., 2017)), pruning of connections (Han et al., 2016, 2015; Guo et al., 2016)

²<https://github.com/tdmeeste/SparseSeqModels>

and low rank approximations (Jaderberg et al., 2014; Tai et al., 2016). Regularization and pruning often lead to mostly random connectivity, and therefore to irregular memory accesses, with little practical effect in terms of hardware speedup. Low rank approximations are structured and thus do achieve speedups, with as notable examples the works of Wen et al. (2016) and Lebedev and Lepitsky (2016).

Whereas above-cited papers specifically explored convolutional networks, our work focuses on recurrent neural networks (RNNs). Similar ideas have been applied there, e.g., see Lu et al. (2016) for a systematic study of various new compact architectures for RNNs, including low-rank models, parameter sharing mechanisms and structured matrices. Also pruning approaches have been shown to be effective for RNNs, e.g., by Narang et al. (2017). Notably, in the area of audio synthesis, Kalchbrenner et al. (2018) showed that large sparse networks perform better than small dense networks. Their sparse models were obtained by pruning, and importantly, a significant speedup was achieved through an efficient implementation.

For the domain of natural language processing (NLP), recent work by Wang et al. (2016) provides an overview of sparse learning approaches, and in particular noted that “application of sparse coding in language processing is far from extensive, when compared to speech processing”. Our current work attempts to further fill that gap. In contrast to aforementioned approaches (that either rely on inducing sparseness starting from a denser model, or rather indirectly try to impose sparseness by enforcing constraints), we explore ways to predefine sparseness.

In the future, we aim to design models where predefined sparseness will allow using very large representation sizes at a limited computational cost. This could be interesting for training models on very large datasets (Chelba et al., 2013; Shazeer et al., 2017), or for more complex applications such as joint or multi-task prediction scenarios (Miwa and Bansal, 2016; Bekoulis et al., 2018; Hashimoto et al., 2017).

3 Predefined Sparseness in RNNs

Our first objective is designing a recurrent network cell with fewer trainable parameters than a standard cell, with given input dimension i and hidden state size h . In Section , we describe one

way to do this, while still allowing the use of fast RNN libraries in practice. This is illustrated for the task of language modeling in Section .

3.1 Sparse RNN Composed of Dense RNNs

The weight matrices in RNN cells can be divided into input-to-hidden matrices $\mathbf{W}_{hi} \in \mathbb{R}^{h \times i}$ and hidden-to-hidden matrices $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ (assuming here the output dimension corresponds to the hidden state size h), adopting the terminology used in (Goodfellow et al., 2016). A *sparse* RNN cell can be obtained by introducing sparseness in \mathbf{W}_{hh} and \mathbf{W}_{hi} . Note that our experiments make use of the Long Short-Term Memory (LSTM) cell (Hochreiter and Schmidhuber, 1997), but our discussion should hold for any type of recurrent network cell. For example, an LSTM contains 4 matrices \mathbf{W}_{hh} and \mathbf{W}_{hi} , whereas the Gated Recurrent Unit (GRU) (Chung et al., 2014) only has 3.

We first propose to organize the hidden dimensions in several disjoint groups, i.e. N segments with lengths s_n ($n = 1, \dots, N$), with $\sum_n s_n = h$.

We therefore reduce \mathbf{W}_{hh} to a block-diagonal matrix. For example, a uniform segmentation would reduce the number of trainable parameters in \mathbf{W}_{hh} to a fraction $1/N$. Figure 1 illustrates an example \mathbf{W}_{hh} for $N = 3$. One would expect that this simplification has a significant regularizing effect, given that the number of possible interactions between hidden dimensions is strongly reduced. However, our experiments (see Section) show that a larger sparse model may still be more expressive than its dense counterpart with the same number of parameters. Yet, Merity et al. (2017) showed that applying weight dropping (i.e., DropConnect, Wan et al. (2013)) in an LSTM’s \mathbf{W}_{hh} matrices has a stronger positive effect on language models than other ways to regularize them. Sparsifying \mathbf{W}_{hh} upfront can hence be seen as a similar way to avoid the model’s ‘over-expressiveness’ in its recurrent weights.

As a second way to sparsify the RNN cell, we propose to not provide all hidden dimensions with explicit access to each input dimension. In each row of \mathbf{W}_{hi} we limit the number of trainable parameters to a fraction $\gamma \in]0, 1]$. Practically, we choose to organize the γi trainable parameters in each row within a window that gradually moves from the first to the last input dimension, when advancing in the hidden (i.e., row) dimension. Furthermore, we segment the hidden dimension of \mathbf{W}_{hi} according to the segmentation of \mathbf{W}_{hh} , and

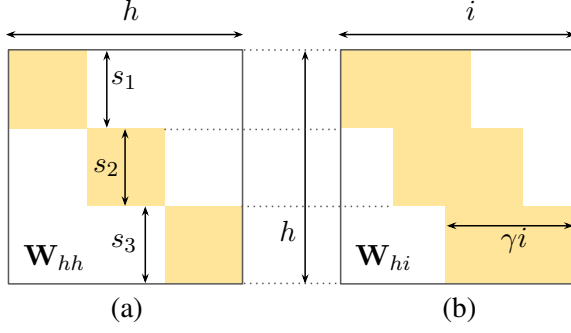


Figure 1: Predefined sparseness in hidden-to-hidden (\mathbf{W}_{hh}) and input-to-hidden (\mathbf{W}_{hi}) matrices in RNNs. Trainable parameters (yellow) vs. zeros (white).

move the window of γi trainable parameters discretely per segment, as illustrated in Fig. 1(b).

Because of the proposed practical arrangement of sparse and dense blocks in \mathbf{W}_{hh} and \mathbf{W}_{hi} , the sparse RNN cell is equivalent to a composition of smaller dense RNN’s operating in parallel on (partly) overlapping input data segments, with concatenation of the individual hidden states at the output. This will be illustrated at the end of Section . As a result, fast libraries like CuDNN (Chetlur et al., 2014) can be used directly. Further research is required to investigate the potential benefit in terms of speed and total cell capacity, of physically distributing computations for the individual dense recurrent cells.

Note that this is only possible because of the initial requirement that the output dimensions are divided into disjoint segments. Whereas inputs can be shared entirely between different components, joining overlapping segments in the h dimension would need to be done within the cell, before applying the gating and output non-linearities. This would make the proposed model less interesting for practical use.

We point out two special cases: (i) dense \mathbf{W}_{hi} matrices ($\gamma = 1$) lead to N parallel RNNs that share the inputs but with separate contributions to the output, and (ii) organizing \mathbf{W}_{hi} as a block matrix (e.g., $\gamma = 1/N$ for N same-length segments), leads to N isolated parallel RNNs. In the latter case, the reduction in trainable parameters is highest, for a given number of segments, but there is no more influence from any input dimension in a given segment to output dimensions in non-corresponding segments. We recommend option (i) as the most rational way to apply our ideas: the

sparse RNN output is a concatenation of individual outputs of a number of RNN components connected in parallel, all sharing the entire input.

3.2 Language Modeling with Sparse RNNs

We apply predefined sparse RNNs to language modeling. Our baseline approach is the AWD-LSTM model introduced by Merity et al. (2017). The recurrent unit consists of a three-layer stacked LSTM (Long Short-Term Memory network (Hochreiter and Schmidhuber, 1997)), with 400-dimensional inputs and outputs, and intermediate hidden state sizes of 1150. Since the vocabulary contains only 10k words, most trainable parameters are in the recurrent layer (20M out of a total of 24M). In order to cleanly measure the impact of predefined sparseness in the recurrent layer, we maintain the original word embedding layer dimensions, and sparsify the recurrent layer.³ In this example, we experiment with increasing dimensions in the recurrent layer while maintaining the number of trainable parameters, whereas in Section we increase sparseness while maintaining dimensions.

Specifically, each LSTM layer is made sparse in such a way that the hidden dimension 1150 is increased by a factor 1.5 (chosen *ad hoc*) to 1725, but the embedding dimensions and total number of parameters remain the same (within error margins from rounding to integer dimensions for the dense blocks). We use uniform segments. The number of parameters for the middle LSTM layer can be calculated as:⁴

params. LSTM layer 2

$$= 4(h_d i_d + h_d^2 + 2h_d) \quad (dense)$$

$$= 4N(\frac{h_s}{N}\gamma i_s + \frac{h_s^2}{N^2} + 2\frac{h_s}{N}) \quad (sparse)$$

in which the first expression represents the general case (e.g., the *dense* case has input and state sizes $i_d = h_d = 1150$), and the second part is the *sparse* case composed of N parallel LSTMs with input size γi_s , and state size h_s/N (with

³Alternative models could be designed for comparison, with modifications in both the embedding and output layer. Straightforward ideas include an ensemble of smaller independent models, or a mixture-of-softmaxes output layer to combine hidden states of the parallel LSTM components, inspired by (Yang et al., 2017).

⁴This follows from an LSTM’s 4 \mathbf{W}_{hh} and 4 \mathbf{W}_{hi} matrices, as well as bias vectors. However, depending on the implementation the equations may differ slightly in the contribution from the bias terms. We assume the standard PyTorch implementation (Paszke et al., 2017).

	finetune	test perplexity
(Merity et al., 2017)	no	58.8
baseline	no	58.8 ± 0.3
sparse LSTM	no	57.9 ± 0.3
(Merity et al., 2017)	yes	57.3
baseline	yes	56.6 ± 0.2
sparse LSTM	yes	57.0 ± 0.2

Table 1: Language modeling for PTB (mean \pm stdev).

$i_s = h_s = 1725$). *Dense* and *sparse* variants have the same number of parameters for $N = 3$ and $\gamma = 0.555$. These values are obtained by identifying both expressions. Note that the equality in model parameters for the dense and sparse case holds only approximately due to rounding errors in (γi_s) and (h_s/N) .

Figure 1 displays \mathbf{W}_{hh} and \mathbf{W}_{hi} for the middle layer, which has close to 11M parameters out of the total of 24M in the whole model. A dense model with hidden size $h = 1725$ would require 46M parameters, with 24M in the middle LSTM alone.

Given the strong hyperparameter dependence of the AWD-LSTM model, and the known issues in objectively evaluating language models (Melis et al., 2017), we decided to keep all hyperparameters (i.e., dropout rates and optimization scheme) as in the implementation from Merity et al. (2017)⁵, including the weight dropping with $p = 0.5$ in the sparse \mathbf{W}_{hh} matrices. Table 1 shows the test perplexity on a processed version (Mikolov et al., 2010) of the Penn Treebank (PTB) (Marcus et al., 1993), both with and without the ‘finetune’ step⁶, displaying mean and standard deviation over 5 different runs. Without finetuning, the sparse model consistently performs around 1 perplexity point better, whereas after finetuning, the original remains slightly better, although less consistently so over different random seeds. We observed that the sparse model overfits more strongly than the baseline, especially during the finetune step. We hypothesize that the regularization effect of *a priori* limiting interac-

⁵Our implementation extends <https://github.com/salesforce/awd-lstm-lm>.

⁶The ‘finetune’ step indicates hot-starting the Averaged Stochastic Gradient Descent optimization once more, after convergence in the initial optimization step (Merity et al., 2017).

tions between dimensions does not compensate for the increased expressiveness of the model due to the larger hidden state size. Further experimentation, with tuned hyperparameters, is needed to determine the actual benefits of predefined sparseness, in terms of model size, resulting perplexity, and sensitivity to the choice of hyperparameters.

4 Sparse Word Embeddings

Given a vocabulary with V words, we want to construct vector representations of length k for each word such that the total number of parameters needed (i.e., non-zero entries), is smaller than kV . We introduce one way to do this based on word frequencies (Section), and present part-of-speech tagging experiments (Section).

4.1 Word-Frequency based Embedding Size

Predefined sparseness in word embeddings amounts to deciding which positions in the word embedding matrix $\mathbf{E} \in \mathbb{R}^{V \times k}$ should be fixed to zero, prior to training. We define the fraction of trainable entries in \mathbf{E} as the embedding density δ_E . We hypothesize that rare words can be represented with fewer parameters than frequent words, since they only appear in very specific contexts. This will be investigated experimentally in Section . Word occurrence frequencies have a typical Zipfian nature (Manning et al., 2008), with many rare and few highly frequent terms. Thus, representing the long tail of rare terms with short embeddings should greatly reduce memory requirements.

In the case of a low desired embedding density δ_E , we want to save on the rare words, in terms of assigning trainable parameters, and focus on the fewer more popular words. An exponential decay in the number of words that are assigned longer representations is one possible way to implement this. In other words, we propose to have the number of words that receive a trainable parameter at dimension j decrease with a factor α^j ($\alpha \in]0, 1]$). For a given fraction δ_E , the parameter α can be determined from requiring the total number of non-zero embedding parameters to amount to a given fraction δ_E of all parameters:

$$\# \text{ embedding params.} = \sum_{j=0}^{k-1} \alpha^j V = \delta_E k V$$

and numerically solving for α .

Figure 2 gives examples of embedding matrices with varying δ_E . For a vocabulary of 44k terms

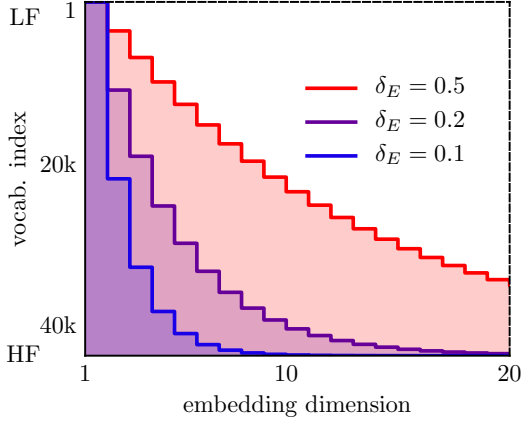


Figure 2: Visualization of sparse embedding matrices for different densities δ_E (with $k = 20$). Colored region: non-zero entries. Rows represent word indices, sorted from least frequent (LF) to highly frequent (HF).

and maximum embedding length $k = 20$, the density $\delta_E = 0.2$ leads to 25% of the words with embedding length 1 (corresponding $\alpha = 0.75$), only 7.6% with length of 10 or higher, and with the maximum length 20 for only the 192 most frequent terms. The particular configurations shown in Fig. 2 are used for the experiments in Section .

In order to set a minimum embedding length for the rarest words, as well as for computational efficiency, we note that this strategy can also be applied on M bins of embedding dimensions, rather than per individual dimensions. The width of the first bin then indicates the minimum embedding length. Say bin m has width κ_m (for $m = 0, \dots, M - 1$, and $\sum_m \kappa_m = k$). The multiplicative decay factor α can then be obtained by solving

$$\delta_E = \frac{1}{k} \sum_{m=0}^{M-1} \kappa_m \alpha^m, \quad (1)$$

while numerically compensating for rounding errors in the number $V\alpha^m$ of words that are assigned trainable parameters in the m^{th} bin.

4.2 Part-of-Speech Tagging Experiments

We now study the impact of sparseness in word embeddings, for a basic POS tagging model, and report results on the PTB Wall Street Journal data. We embed 43,815 terms in 20-dimensional space, as input for a BiLSTM layer with hidden state size 10 for both forward and backward directions. The concatenated hidden states go into a fully

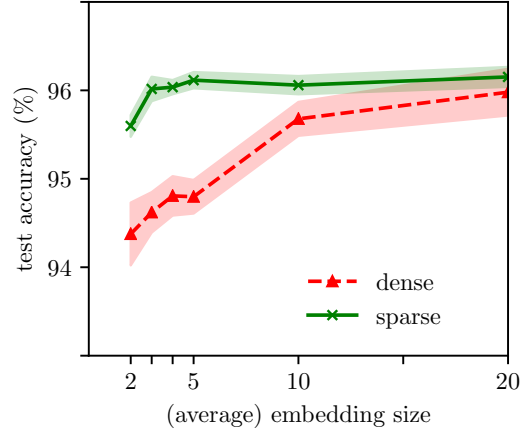


Figure 3: POS tagging accuracy on PTB data: dense (red) vs. sparse (green). X-axis: embedding size k for the dense case, and average embedding size (or $20 \delta_E$) for the sparse case. Shaded bands indicate *stdev* over 4 randomly seeded runs.

connected layer with tanh non-linearity (down to dimension 10), followed by a *softmax* classification layer with 49 outputs (i.e., the number of POS tags). The total number of parameters is 880k, of which 876k in the embedding layer. Although character-based models are known to outperform pure word embedding based models (Ling et al., 2015), we wanted to investigate the effect of sparseness in word embeddings, rather than creating more competitive but larger or complex models, risking a smaller resolution in the effect of changing individual building blocks. To this end we also limited the dimensions, and hence the expressiveness, of the recurrent layer.⁷ Our model is similar to but smaller than the ‘word lookup’ baseline by Ling et al. (2015).

Figure 3 compares the accuracy for variable densities δ_E (for $k = 20$) vs. different embedding sizes (with $\delta_E = 1$). For easily comparing sparse and dense models with the same number of embedding parameters, we scale δ_E , the x-axis for the sparse case, to the average embedding size of $20 \delta_E$.

Training models with shorter dense embeddings appeared more difficult. In order to make a fair comparison, we therefore tuned the models over a

⁷With LSTM state sizes of 50, the careful tuning of dropout parameters gave an accuracy of 94.7% when reducing the embedding size to $k = 2$, a small gap compared to 96.8% for embedding size 50. The effect of larger sparse embeddings was therefore much smaller in absolute value than the one visualized in Fig. 3, because of the much more expressive recurrent layer.

range of regularization hyperparameters, provided in Table 2.

We observe that the sparse embedding layer allows lowering the number of parameters in \mathbf{E} down to a fraction of 15% of the original amount, with little impact on the effectiveness, provided \mathbf{E} is sparsified rather than reduced in size. The reason for that is that with sparse 20-dimensional embeddings, the BiLSTM still receives 20-dimensional inputs, from which a significant subset only transmits signals from a small set of frequent terms. In the case of smaller dense embeddings, information from all terms is uniformly present over fewer dimensions, and needs to be processed with fewer parameters at the encoder input.

Finally, we verify the validity of our hypothesis from Section that frequent terms need to be embedded with more parameters than rare words. Indeed, one could argue in favor of the opposite strategy. It would be computationally more efficient if the terms most often encountered had the smallest representation. Also, stop words are the most frequent ones but are said to carry little information content. However, Table 3 confirms our initial hypothesis. Applying the introduced strategy to sparsify embeddings on randomly ordered words (‘no sorting’) leads to a significant decrease in accuracy compared to the proposed sorting strategy (‘up’). When the most frequent words are encoded with the shortest embeddings (‘down’ in the table), the accuracy goes down even further.

5 Learning To Recite

From the language modeling experiments in Section , we hypothesized that an RNN layer becomes more expressive, when the dense layer is replaced by a larger layer with predefined sparseness and the same number of model parameters. In this section, we design an experiment to further investigate this claim. One way of quantifying an RNN’s capacity is in measuring how much information it can memorize. We name our experimental setup *learning to recite*: we investigate to what extent dense vs. sparse models are able to learn an entire corpus by heart in order to recite it afterwards. We note that this toy problem could have interesting applications, such as the design of neural network components that keep entire texts or even knowledge bases available for later retrieval,

encoded in the component’s weight matrices.⁸

5.1 Experimental Results

The initial model for our *learning to recite* experiment is the baseline language model used in Section (Merity et al., 2017), with the PTB data. We set all regularization parameters to zero, to focus on memorizing the training data. During training, we measure the ability of the model to correctly predict the next token at every position in the training data, by selecting the token with highest predicted probability. When the model reaches an accuracy of 100%, it is able to recite the entire training corpus. We propose the following optimization setup (tuned and tested on dense models with different sizes): minibatch SGD (batch size 20, momentum 0.9, and best initial learning rate among 5 or 10). An exponentially decaying learning rate factor (0.97 every epoch) appeared more suitable for memorization than other learning rate scheduling strategies, and we report the highest accuracy in 150 epochs.

We compare the original model (in terms of network dimensions) with versions that have less parameters, by either reducing the RNN hidden state size h or by sparsifying the RNN, and similarly for the embedding layer. For making the embedding matrix sparse, $M = 10$ equal-sized segments are used (as in eq. 1). Table 4 lists the results. The dense model with the original dimensions has 24M parameters to memorize a sequence of in total ‘only’ 930k tokens, and is able to do so. When the model’s embedding size and intermediate hidden state size are halved, the number of parameters drops to 7M, and the resulting model now makes 67 mistakes out of 10k predictions. If h is kept, but the recurrent layers are made sparse to yield the same number of parameters, only 5 mistakes are made for every 10k predictions. Making the embedding layer sparse as well introduces new errors. If the dimensions are further reduced to a third the original size, the memorization capacity goes down strongly, with less than 4M trainable parameters. In this case, sparsifying both the recurrent and embedding layer yields the best result, whereas the dense model works better than the model with sparse RNNs only. A possible explanation for that is the strong sparseness in the RNNs. For example, in the middle layer only 1

⁸It is likely that recurrent networks are not the best choice for this purpose, but here we only wanted to measure the LSTM-based model’s capacity to memorize with and without predefined sparseness.

hyperparameter	value(s)
optimizer	Adam (Kingma and Ba, 2015)
learning rate	0.001
epochs	50
word level embedding dropout †	[0.0, 0.1, 0.2]
variational embedding dropout †	[0.0, 0.1, 0.2, 0.4]
DropConnect on \mathbf{W}_{hh} †	[0.0, 0.2, 0.4]
batch size	20

Table 2: Hyperparameters for POS tagging model (†as introduced in (Merity et al., 2017)). A list indicates tuning over the given values was performed.

	$\delta_E = 1.0$	$\delta_E = 0.25$	$\delta_E = 0.1$
# params. (E; all)	876k; 880k	219k; 222k	88k ; 91k
up		96.1 \pm 0.1	95.6 \pm 0.1
no sorting	96.0 \pm 0.3	94.3 \pm 0.4	93.0 \pm 0.3
down		89.8 \pm 2.2	90.6 \pm 0.5

Table 3: Impact of vocabulary sorting on POS accuracy with sparse embeddings: up vs. down (most frequent words get longest vs. shortest embeddings, resp.) or not sorted, for different embedding densities δ_E .

	embeddings		hidden state	# parameters	memorization accuracy (%)
	size k ,	density δ_E	size h		
dense model (orig. dims.)	400	1	1150	24.22M	100.0
dense model (see Fig. 4(a))	200	1	575	7.07M	99.33
sparse RNN (see Fig. 4(b))	200	1	1150	7.07M	99.95
sparse RNN + sparse emb.	400	1/2	1150	7.07M	99.74
dense model	133	1	383	3.59M	81.48
sparse RNN	133	1	1150	3.59M	76.37
sparse RNN + sparse emb.	400	1/3	1150	3.59M	89.98

Table 4: PTB train set memorization accuracies for dense models vs. models with predefined sparseness in recurrent and embedding layers with comparable number of parameters.

out of 10 recurrent connections is non-zero. In this case, increasing the size of the word embeddings (at least, for the frequent terms) could provide an alternative for the model to memorize parts of the data, or maybe it makes the optimization process more robust.

5.2 Visualization

Finally, we provide an illustration of the high-level composition of the recurrent layers in two of the models used for this experiment. Figure 4(a) sketches the stacked 3-layer LSTM network from the ‘dense RNN’ model (see Table 4) with $k = 200$ and $h = 575$. As already mentioned, our

proposed sparse LSTMs are equivalent to a well-chosen composition of smaller dense LSTM components with overlapping inputs and disjoint outputs. This composition is shown in Fig. 4(b) for the model ‘sparse RNN’ (see Table 4), which in every layer has the same number of parameters as the dense model with reduced dimensions.

6 Conclusion and Future Work

This paper introduces strategies to design word embedding layers and recurrent networks with predefined sparseness. Effective sparse word representations can be constructed by encoding less frequent terms with smaller embeddings and vice

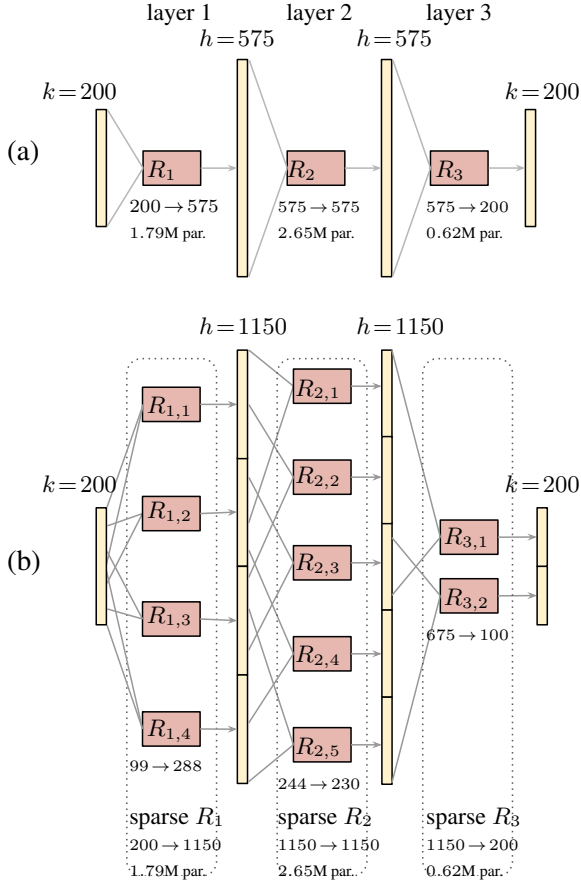


Figure 4: Schematic overview of 3-layer stacked (a) dense vs. (b) sparse LSTMs with the same number of parameters (indicated with ‘par.’). Sparse layers are effectively composed of smaller dense LSTMs. ‘ $R_{i,j}$ ’ indicates component j within layer i , and ‘ $675 \rightarrow 100$ ’ indicates an LSTM component with input size 675 and output size 100.

versa. A sparse recurrent neural network layer can be constructed by applying multiple smaller recurrent cells in parallel, with partly overlapping inputs and concatenated outputs.

The presented ideas can be applied to build models with larger representation sizes for a given number of parameters, as illustrated with a language modeling example. Alternatively, they can be used to reduce the number of parameters for given representation sizes, as investigated with a part-of-speech tagging model.

We introduced ideas on predefined sparseness in sequence models, as well as proof-of-concept experiments, and analysed the memorization capacity of sparse networks in the ‘learning to recite’ toy problem.

More elaborate experimentation is required to

investigate the benefits of predefined sparseness on more competitive tasks and datasets in NLP. For example, language modeling results on the Penn Treebank rely on heavy regularization due to the small corpus. Follow-up work could therefore investigate to what extent language models for large corpora can be trained with limited computational resources, based on predefined sparseness. Other ideas for future work include the use of predefined sparseness for pretraining word embeddings, or other neural network components besides recurrent models, as well as their use in alternative applications such as sequence-to-sequence tasks or in multi-task scenarios.

Acknowledgments

We thank the anonymous reviewers for their time and effort, and the valuable feedback.

References

- Giannis Bekoulis, Johannes Deleu, Thomas Demeester, and Chris Develder. 2018. Joint entity recognition and relation extraction as a multi-head selection problem. *Expert Systems with Applications*, 114:34–45.
- Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. Technical report, Google.
- Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient primitives for deep learning. *arXiv:1410.0759*.
- Junyoung Chung, Çağlar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*. Deep Learning workshop at NIPS 2014.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- Yiwen Guo, Anbang Yao, and Yurong Chen. 2016. Dynamic network surgery for efficient DNNs. In *Proc. 30th International Conference on Neural Information Processing Systems (NIPS 2016)*, NIPS’16, pages 1387–1395.
- Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proc. 4th International Conference on Learning Representations (ICLR 2016)*.
- Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning Both Weights and Connections for

- Efficient Neural Networks. In *Proc. 28th International Conference on Neural Information Processing Systems (NIPS 2015)*, NIPS'15, pages 1135–1143.
- Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsunoda, and Richard Socher. 2017. A joint many-task model: Growing a neural network for multiple nlp tasks. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1923–1933.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. In *Proc. 27th British Machine Vision Conference (BMVC 2014)*. ArXiv: 1405.3866.
- Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aäron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. 2018. Efficient neural audio synthesis. ArXiv: 1802.08435.
- Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, San Diego, USA.
- V. Lebedev and V. Lempitsky. 2016. Fast ConvNets using group-wise brain damage. In *Proc. 29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*, pages 2554–2564.
- Wang Ling, Chris Dyer, Alan W Black, Isabel Trancoso, Ramon Fernandez, Silvio Amir, Luis Marujo, and Tiago Luis. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal. Association for Computational Linguistics.
- Baoyuan Liu, Min Wang, H. Foroosh, M. Tappen, and M. Pensky. 2015. Sparse convolutional neural networks. In *Proc. 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)*, pages 806–814.
- Zhiyun Lu, Vikas Sindhwani, and Tara N. Sainath. 2016. Learning compact recurrent neural networks. In *Proc. 41st IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2016)*.
- Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.
- Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.
- Gábor Melis, Chris Dyer, and Phil Blunsom. 2017. On the state of the art of evaluation in neural language models. In *Proc. 6th International Conference on Learning Representations (ICLR 2017)*.
- Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing LSTM language models. *arXiv:1708.02182*.
- Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048. ISCA.
- Makoto Miwa and Mohit Bansal. 2016. End-to-end relation extraction using LSTMs on sequences and tree structures. In *Proc. 54th Annual Meeting of the Association for Computational Linguistics*, pages 1105–1116.
- Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. 2017. Variational dropout sparsifies deep neural networks. In *Proc. 35th International Conference on Machine Learning (ICML 2017)*. ArXiv: 1701.05369.
- Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. 2017. Exploring sparsity in recurrent neural networks. In *Proc. 5th International Conference on Learning Representations (ICLR 2017)*.
- Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *Proceedings of the Workshop on The future of gradient-based machine learning software and techniques, co-located with the 31st Annual Conference on Neural Information Processing Systems (NIPS 2017)*.
- Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proc. International Conference on Learning Representations (ICLR)*.
- Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, and Weinan E. 2016. Convolutional neural networks with low-rank regularization. In *Proc. 4th International Conference on Learning Representations (ICLR 2016)*. ArXiv: 1511.06067.
- Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. In *Proc. 30th International Conference on International Conference on Machine Learning (ICML 2013)*, pages III–1058–III–1066, Atlanta, GA, USA.
- Dong Wang, Qiang Zhou, and Amir Hussain. 2016. Deep and sparse learning in speech and language processing: An overview. In *Proc. 8th International Conference on (BICS2016)*, pages 171–183. Springer, Cham.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Proc. 30th International Conference on Neural Information Processing Systems (NIPS 2016)*, NIPS’16, pages 2082–2090, USA.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2017. Breaking the softmax bottleneck: A high-rank rnn language model. ArXiv: 1711.03953.