NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

as a manuscript

Ekaterina Lobacheva

DEEP LEARNING ARCHITECTURES ON A LIMITED MEMORY BUDGET

PhD Dissertation Summary
for the purpose of obtaining academic degree
Doctor of Philosophy in Computer Science

Moscow — 2022

The PhD Dissertation was prepared at National Research University Higher School of Economics.

Academic Supervisor: Dmitry P. Vetrov, Candidate of Science, National Research University Higher School of Economics.

1 Introduction

Topic of the thesis

This work studies the problem of obtaining high-quality deep learning models in the presence of memory limitations at the inference stage. Such limitations can arise, for example, when using neural networks in user applications that run on personal computers or smartphones and should not take up too much memory on the devices.

One of the most popular approaches for the considered problem is sparsification of neural network weights during training. In the first part of the work, we develop new sparsification techniques for recurrent neural networks. First, we propose a modification of the Bayesian sparsification approach, Sparse variational dropout [1], taking into account the recurrent structure of the model. Then we develop a new method of structured sparsification for modern gated recurrent architectures, improving compression results both in the Bayesian framework and in standard pruning.

In the second part of the work, we study an ensembling of neural networks and show how it can be used to obtain high-quality models which are small in size. We discover the memory split advantage effect — for a wide range of total model sizes, an ensemble of several small neural networks shows better results than one large neural network. Thus, for a given memory budget, instead of training a single neural network of a given size or a larger neural network with sparsified weights, one can divide the budget and train an ensemble of several smaller networks, obtaining a better quality. We also show that in many cases, the quality of an ensemble behaves as a power law with respect to the number of networks or their size. The discovered power-law behavior allows predicting the quality of large ensembles without training them and the optimal memory split of a fixed budget.

Relevance

Modern deep learning models successfully solve a wide range of problems in computer vision, natural language processing, speech recognition, etc. Latest research [2, 3] shows that large models and large amounts of training data are the key ingredients to the deep learning success. Most of the leading models for various tasks [2, 4, 3, 5] have millions, and sometimes even billions, of parameters, require large amounts of training

data and significant computational resources for training and inference. However, in practical scenarios, resources are often limited, especially during the application on the user side. In this paper, we consider the problem of obtaining high-quality models in the presence of memory limitations at the inference stage. Such limitations may arise, for example, when using neural networks in applications that run on personal computers or smartphones and should not take up too much memory on the devices.

Several classic approaches for this problem are presented in the literature: weight sparsification [6, 1], weight quantization [6], network distillation [7] and weight matrix tensorization [8, 9]. In all of them, a neural network is trained in a specialized way to obtain a model which is small in terms of the number of parameters (or memory size in case of quantization) but achieves the results of a larger model in terms of quality. Each approach applies its own restriction on the training process to compress the model. Weight sparsification introduces a sparsity-inducing regularizer on network weights during training. Such a regularizer makes some weights or weight groups very close to zero, therefore, the model size can be reduced by excluding them from the model after training. Weight quantization constrains the number of possible weight values during training, for example, int8 is used instead of float32 weights. In distillation, a large teacher network is first trained, and then a small student network is trained to emulate the responses of the teacher network. In tensorization, network weight matrices are represented with products of low-rank tensors. All described approaches are actively researched in the literature and can be combined to achieve better results [6, 10, 11]. At the same time, their application to neural network architectures of different structures has its specifics and needs further investigation.

In the first part of the work, we focus on the weight sparsification approach, which shows high results on basic architectures, such as fully connected and convolutional networks. We develop sparsification methods for more complex architectures — recurrent neural networks. Recurrence introduces important specifics into the network and needs to be properly addressed during sparsification. In the forward pass, weights of recurrent layers affect network predictions multiple times, specifically when processing each token of an input sequence. That makes it challenging to understand which model weights are important and which are not, complicating the sparsification procedure. In the first paper, we show how the recurrence can be taken into account and propose an adaptation of an effective Bayesian sparsification method, Sparse variational dropout [1], for recur-

rent neural networks. We then extend the proposed method to additionally sparsify the model's input dictionary and further improve the compression. In the second paper, we propose a structured sparsification method that takes into account the gated structure of modern recurrent architectures, such as LSTM [12]. This method introduces three sparsity levels to the model: it removes some neurons from the network, makes some gates of the remaining neurons constant, and sets some weights of non-constant gates to zero. The proposed technique can be implemented in the Bayesian framework and in standard pruning [13, 14] and improves the sparsification results in both of them. We also conduct a qualitative analysis of the results of both proposed methods and show that the network structure obtained after sparsification is interpretable and highly depends on the task.

In the second part of this work, a more non-standard approach to obtaining high-quality models on a limited memory budget is proposed based on the ensembling of neural networks [15]. In practice, small neural networks trained without specialized compression techniques usually work much worse than the large ones [16, 17]. However, the quality of the neural network may be improved not only by increasing the size of the model but also by taking an ensemble of several models. In the third paper, we analyze how the quality of the neural network ensemble behaves if we increase the number of networks in the ensemble or the size of the networks. The results show that for a wide range of total model sizes, an ensemble of several small networks can perform better than one large network. This is true even for sufficiently small total model sizes, which makes it possible to use an ensembling as an approach to obtain small high-quality models. So, given a fixed memory budget, instead of training a single network of a given size or a larger model with compression, one can split the budget and train an ensemble of several smaller networks and get a quality boost. We also show that ensemble quality behaves as a power law with respect to the number of networks and the size of the networks in a large number of cases. Our results are consistent with the results of Kaplan et al. [16], Rosenfeld et al. [17] on the dependence of the quality of an individual neural network on its size. The discovered power laws allow accurate predictions of the quality of large ensembles without their training and the optimal memory split of a fixed memory budget.

In summary, **the goal of this work** is to develop existing and propose new methods and approaches for obtaining high-quality deep learning models in the presence of memory limitations at the inference stage.

2 Key results and conclusions

Contributions. The main contributions of this work can be summarized as follows:

1. We proposed an adaptation of the Bayesian sparsification method for recurrent neural networks, which allows sparsification of both the weights of the recurrent layers and the vocabulary of the model. The obtained method allows compressing models for language modeling and text classification tasks by tens-hundreds times without a significant quality drop.

2. We proposed a structured sparsification method for recurrent architectures with gates, which introduces hierarchical sparsification into the network by sparsifying weights, gates, and neurons. This method is applicable both in the Bayesian framework and in standard pruning, and improves the level of structural compression (the number of remaining neurons) for language modeling and text classification tasks without a significant quality drop.

3. We studied the behavior of the ensemble quality of convolutional neural networks with respect to the number of networks in the ensemble and the size of the networks. We discovered that the ensemble quality behaves as a power law in many situations and also discussed when this is not the case. We then conducted a thorough analysis of the discovered power laws and their parameters.

4. We revealed the memory split advantage effect: for a wide range of the total model sizes, an ensemble of several small neural networks shows better results than one large neural network.

5. Based on the discovered power laws, we proposed a method for predicting the optimal memory split, i.e. the optimal number of networks in the ensemble, for a fixed memory budget.

Theoretical and practical significance. This work proposes new methods and approaches to the compression of neural network models, hence, simplifying the usage of these models in practical applications on user devices with limited memory resources. The conducted empirical and theoretical study of the behavior of the ensemble quality with respect to the number of networks and their sizes not only allows us to look at ensembling as one of the approaches to obtaining small high-quality models but also generally shows

that ensembling should be considered as one of the methods for increasing the size of the model, along with increasing the width or depth of networks.

Key aspects/ideas to be defended:

1. The Bayesian weight sparsification method for recurrent neural networks.
2. The structured sparsification method for gated recurrent neural networks, applicable in the Bayesian framework and standard pruning.
3. Power-law behavior of the ensemble quality of convolutional neural networks with respect to the number of networks and their size, and the study of these power laws.
4. The memory split advantage effect for neural network ensembles on a fixed memory budget.
5. The prediction method for the optimal memory split of a fixed memory budget, based on the discovered power laws.

Personal contribution. In the first two papers, the methods were developed jointly with Nadezhda Chirkova. The experimental results on the language modeling task were obtained by the author of this work, while the results on the text classification task were obtained by Nadezhda Chirkova. Dmitry Vetrov provided scientific guidance for the project and helped with the expertise in the subject area. Alexander Markovich provided technical assistance in the experiments for the second paper.

In the third paper, the author obtained all empirical results on power laws, as well as on their application to predicting the quality of large ensembles and the optimal memory splits for a fixed memory budget. Theoretical results on power laws were obtained jointly with Maxim Kodryan. The results on the memory split advantage effect were obtained jointly with Nadezhda Chirkova. Dmitry Vetrov provided scientific guidance for the project and helped with expertise in the subject area.

Publications and probation of the work

First-tier publications

1. Nadezhda Chirkova*, **Ekaterina Lobacheva***, Dmitry Vetrov. Bayesian Compression for Natural Language Processing. In Proceedings of Conference on Empirical Methods in Natural Language Processing, 2018 (EMNLP 2018). Pages 2910-2915. CORE A conference.

2. **Ekaterina Lobacheva***, Nadezhda Chirkova*, Alexander Markovich, Dmitry Vetrov. Structured Sparsification of Gated Recurrent Neural Networks. In Proceedings of the AAAI Conference on Artificial Intelligence, 2020 (AAAI 2020). Vol. 34, No. 04, pages 4989-4996. CORE A* conference.

3. **Ekaterina Lobacheva**, Nadezhda Chirkova, Maxim Kodryan, Dmitry Vetrov. On Power Laws in Deep Ensembles. In Advances in Neural Information Processing Systems, 2020 (NeurIPS 2020). Vol. 33, pages 2375-2385. CORE A* conference.

* — authors with equal contribution.

Reports at conferences, workshops and seminars

1. Workshop on Learning to Generate Natural Language at ICML, 10 August 2017. Topic: "Bayesian Sparsification of Recurrent Neural Networks".

2. Conference on Empirical Methods in Natural Language Processing, 3 November 2018. Topic: "Bayesian Compression for Natural Language Processing".

3. Workshop on Compact Deep Neural Networks with industrial applications at NeurIPS, 7 December 2018. Topic: "Bayesian Sparsification of Gated Recurrent Neural Networks".

4. Workshop on Context and Compositionality in Biological and Artificial Neural Systems at NeurIPS, 14 December 2019. Topic: "Structured Sparsification of Gated Recurrent Neural Networks".

5. AAAI Conference on Artificial Intelligence, 11 February 2020. Topic: "Structured Sparsification of Gated Recurrent Neural Networks".

6. Workshop on Uncertainty and Robustness in Deep Learning at ICML, 17 July 2020. Topic: "On Power Laws in Deep Ensembles".

7. Seminar of the Bayesian methods research group, 16 October 2020. Topic: "On Power Laws in Deep Ensembles".

8. Seminar of the Faculty of Computer Science in Voronovo, 18 November 2020. Topic: "On Power Laws in Deep Ensembles".

9. Conference on Neural Information Processing Systems, 10 December 2020. Topic: "On Power Laws in Deep Ensembles".

Volume and structure of the work. The thesis contains an introduction, contents of publications and a conclusion. The full volume of the thesis is 64 pages.

3 Sparsification of recurrent neural networks

Recurrent neural networks are usually used for the processing of sequential data, such as text, time series, etc. A recurrent architecture receives a sequence of tokens $x = [x_0, \dots, x_T]$ as an input, processes them sequentially, and outputs a prediction $\hat{y}$, which can be either a scalar in the case of a sequence classification task or a vector in the case of the sequence generation task. For example, in the case of a sequence classification task, a forward pass for a simple recurrent neural network would look as follows:

$$\begin{aligned}
&\text{embedding layer} : \tilde{x}_t = w^e_{x_t}, \\
&\text{recurrent layer} : h_{t+1} = \sigma(W^h h_t + W^x \tilde{x}_{t+1} + b^r), \\
&\text{fully connected last layer} : \hat{y} = \text{softmax}(W^d h_T + b^d),
\end{aligned}$$

where W and b denote trainable weight matrices and bias vectors respectively, and h_t denotes a vector of hidden neurons after processing the input token x_t.

Because of the recurrence, the network weights affect the predictions for an object not once but multiple times, specifically during the processing of each token of the input sequence. That makes it difficult to understand which model weights are important and which are not, complicating the sparsification procedure.

3.1 Bayesian sparsification of recurrent neural networks

Most of the previously existing methods for recurrent neural networks sparsification are heuristic and require careful selection of hyperparameters [13, 14] or a network compression structure [18, 19]. In this section, we propose an adaptation of a theoretically-grounded sparsification method, Sparse variational dropout (SparseVD) [1], to the recurrent neural networks, taking into account the specifics of recurrent structure. We also show how to extend the proposed method to sparsification of the model's vocabulary.

In this section, we denote all the weights of the recurrent neural network, except for the bias vectors, as ω, while one weight from this set as w_{ij}. The bias vectors are denoted by a separate variable B since they are not sparsified in the proposed method.

Weight sparsification

To sparsify a neural network in a Bayesian framework, one should treat it as a Bayesian neural network and obtain a posterior distribution for weights during training instead of a point estimate. Following [20, 1], we use a fully factorized log-uniform prior distribution for the weights $p(\omega) = \prod_{w_{ij}\in\omega} p(w_{ij}), \quad p(w_{ij}) \propto 1/|w_{ij}|$ and approximate the posterior distribution with a fully factorized normal distribution $q(w|\theta,\sigma) = \prod_{w_{ij}\in\omega} \mathcal{N}\big(w_{ij}|\theta_{ij},\sigma_{ij}^2\big)$.

To obtain the approximation to the posterior distribution, we optimize the variational lower bound [1]:

$$-\sum_{i=1}^{N}\int q(\omega|\theta,\sigma)\log p(y^i|x_0^i,\ldots,x_T^i,\omega,B)d\omega+ \\ +\sum_{w_{ij}\in\omega} KL(q(w_{ij}|\theta_{ij},\sigma_{ij})||p(w_{ij}))\to\min_{\theta,\sigma,B}. \tag{1}$$

Here the first term corresponds to the task-specific loss and is approximated with one sample from $q(\omega|\theta,\sigma)$. The second term is a regularizer that induces sparsity. It can be closely approximated analytically with a function depending on θ and σ [1].

To obtain an unbiased estimate of the gradient of the optimized functional, we use the reparametrization trick [21]:

$$w_{ij} = \theta_{ij} + \sigma_{ij}\epsilon_{ij}, \quad \epsilon_{ij}\sim\mathcal{N}(\epsilon_{ij}|0,1) \tag{2}$$

Recurrent neural networks share the weights between different time steps. Hence, for the correct implementation of the described approach, the same sample of weights should be used for all time steps t when calculating the likelihood $p(y^i|x_0^i,\ldots,x_T^i,\omega,B)$ [22, 23].

Moreover, in the case of recurrent neural networks, the local reparametrization trick [20, 1] is not applicable for some weights. In local reparametrization, neuron pre-activations are sampled instead of individual weights, for example:

$$(W^x x_t)_i = \sum_j \theta_{ij}^x x_{tj} + \epsilon_i\sqrt{\sum_j(\sigma_{ij}^x)^2 x_{tj}^2}. \tag{3}$$

Local reparametrization is not applicable to shared weight matrices that are used in more than one time step because of the tied sampling of such matrices at different time steps.

For the hidden-to-hidden matrix W^h, the linear combination $(W^h h_t)$ is not normally distributed, since h_t depends on W^h from the previous step. As a result, the rule about

the sum of independent normal distributions with constant coefficients is not applicable. In practice, recurrent networks with local reparametrization for $(W^h h_t)$ cannot be trained properly.

For the input-to-hidden matrix W^x, the linear combination $(W^x x_t)$ is normally distributed, but sampling the same matrix W^x for all time steps and sampling the same noise ϵ_i for all time steps are not equivalent. In practice, recurrent networks with local reparametrization for $(W^x x_t)$ can perform just as well as the ones without local reparametrization, and their training even converges faster in some cases.

As a result, the training looks as follows. On the forward pass through the network, the weights ω are sampled using the formula (2), and then the recurrent network is applied in a standard way. On the backward pass, the gradients of the loss with respect to $\theta, \log\sigma, B$ are calculated, and then they are used for optimization.

At the inference stage, we use a deterministic network with the mean θ as the weights. Because of the regularizer (the second term in (1)), many elements of θ converge to low absolute values. As a result, we can sparsify the network by zeroing out the weights with the signal-to-noise ratio below a certain threshold: $\theta_{ij}^2/\sigma_{ij}^2 < \tau$ (the same is done in [1]).

Vocabulary sparsification

One of the advantages of Bayesian sparsification methods is that they can be easily generalized to sparsify any group of weights without significantly complicating the learning procedure. To do so, new stochastic multiplicative weights for such groups are usually introduced [24]. In this work, we propose to additionally sparsify the model input vocabulary and, as a result, the matrix of weights of the input layer. This matrix is a significant part of the model in terms of the occupied memory for many tasks.

To sparsify the input vocabulary, we introduce multiplicative stochastic weights $z \in \mathbb{R}^V$ for the words in the vocabulary (here, V denotes the size of the vocabulary). The forward path through the network then looks as follows:

1. sample vector z^i from the current approximation of the posterior distribution for each input sequence x^i in the mini-batch;
2. for each token x_t^i from the sequence x^i multiply its one-hot encoding vector by z^i;
3. continue the forward pass in the standard regime.

Table 1: Results of Bayesian sparsification of weights and input vocabulary. Compression is equal to $|w|/|w \neq 0|$. The penultimate column shows the number of elements left in the model's vocabulary, and the last column shows the number of remaining neurons in the embedding and recurrent layers (the embedding layer is used only in the case of a text classification problem). In the case of language modeling, the quality on the validation and test sets is given.

Task	Method	Quality	Compression	Vocabulary	Neurons
	Original	84.1	1x	20000	$300 - 128$
IMDb	SparseVD	**85.1**	1135x	4611	$16 - 17$
Accuracy %	SparseVD-Voc	83.6	**18792x**	**292**	$\mathbf{1 - 8}$
	Original	**90.6**	1x	20000	$300 - 512$
AGNews	SparseVD	88.8	322x	5727	$179 - 56$
Accuracy %	SparseVD-Voc	89.2	**469x**	**2444**	$\mathbf{127 - 32}$
	Original	$1.499 - 1.454$	1x	50	1000
Char PTB	SparseVD	$1.472 - 1.429$	**7.9x**	50	**431**
Bits-per-char	SparseVD-Voc	$\mathbf{1.458 - 1.417}$	6.0x	**46**	510
	Original	$135.6 - 129.3$	1x	10000	256
Word PTB	SparseVD	$\mathbf{115.0 - 109.2}$	**22.1x**	9990	**156**
Perplexity	SparseVD-Voc	$126.0 - 120.2$	19.3x	**3164**	209

During the training, we work with the weights z in the same way as with the weights W. We use a fully-factorized log-uniform prior and approximate the posterior distribution in a family of fully-factorized normal distributions with trainable means and variances. After training, we set elements of z with a low signal-to-noise ratio to zero and remove the corresponding vocabulary elements and input layer matrix columns from the model.

Empirical evaluation

We perform experiments with LSTM recurrent architecture [12] on the problems of character-level and word-level language modeling (dataset — Penn Treebank [25]) and text classification (datasets — IMDb [26] and AGNews [27]). Table 1 shows a comparison of three approaches in terms of quality and compression: baseline training without sparsification (Original), training with the proposed weight sparsification technique (SparseVD), and training with both weight and vocabulary sparsification (SparseVD-Voc).

The results show that the proposed weight sparsification technique leads to a high compression rate without a significant quality drop. Vocabulary sparsification substantially

improves the compression rate in the case of classification problems since, usually, only a small subset of relevant words is needed for the correct classification of texts. For example, the model, trained to classify the film reviews from the IMDb dataset into positive and negative, has only 292 words in the vocabulary, many of which have a pronounced positive or negative connotation (great, hilarious, terrible, horrible, etc.). When considering character-level language modeling, vocabulary sparsification does not improve the results as all letters are important in this task. With word-level language modeling, more than half of the words are excluded from the model. However, the quality of the model suffers significantly from this. Thus, the proposed weight sparsification technique is applicable to all the considered tasks, while vocabulary sparsification has a positive effect only in cases with a text classification problem.

3.2 Structured sparsification of gated recurrent neural networks

Previously existing sparsification methods for recurrent neural networks aim at sparsification either at the level of individual weights Narang et al. [13], See et al. [28] or at the level of hidden neurons [14]. However, most modern recurrent neural networks used in practice have gated architecture. For example, during the forward pass through the LSTM layer [12] for the input token x_t, one first computes the values of three gates, the input gate i, the output gate o, and the forget gate f, as well as the information flow g (which we will also call a gate for generality), and only after that uses their values to compute the output of the layer h:

$$\begin{aligned} i_t &= sigm(W_i^x x_t + W_i^h h_{t-1} + b_i) & f_t &= sigm(W_f^x x_t + W_f^h h_{t-1} + b_f) \\ g_t &= tanh(W_g^x x_t + W_g^h h_{t-1} + b_g) & o_t &= sigm(W_o^x x_t + W_o^h h_{t-1} + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t & h_t &= o_t \odot tanh(c_t) \end{aligned} \tag{4}$$

In this section, we propose a structured sparsification method that takes into account the gate structure of networks and is applicable both in the Bayesian framework and in standard pruning.

Structured sparsification method

The main idea of the method is to sparsify intermediate elements, gates, in addition to sparsification of individual weights and entire neurons. On the one hand, this approach

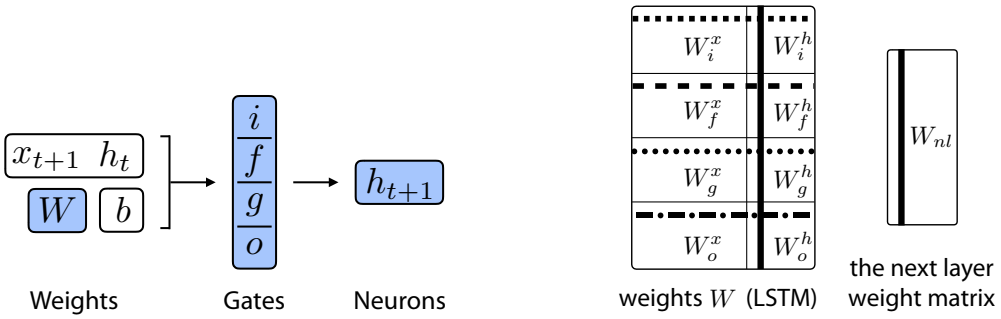

Figure 1: Structured sparsification method for gated recurrent neural networks. Left: proposed three levels of sparsity — weights, gates and neurons (highlighted in blue). Right: proposed weight groups for the LSTM architecture. Different groups are shown using different line types — horizontal dotted lines correspond to groups of four gates, and two vertical solid lines correspond to a group of a neuron.

allows turning off gates, simplifying the structure of some neurons. On the other hand, the hierarchical structure of sparsification, weights-gates-neurons (see Figure 1, left), improves the sparsity level of neurons, which is the most important in practice: sparsification of individual weights helps to sparsify gate pre-activations, i.e. making the gates constant, and this, in turn, helps to sparsify the neurons.

The described idea can be applied to any gated architecture. We consider applying it to the LSTM architecture as an example. The procedure for computing gate values in this architecture can be represented as a fully-connected neural network layer that takes the vector $[x_t, h_{t-1}]$ as input and transforms it by multiplying by the combined weight matrix W and adding the combined bias vector $b = [b_i, b_f, b_g, b_o]$. The structure of the weight matrix W is shown in Figure 1, right.

Sparsification of structured elements, such as gates or neurons, implies removing or zeroing out the weight groups associated with them. We sparsify only elements of weight matrices and not the bias vectors as the latter do not take up much memory. For each gate element, there is an associated row of the weight matrix W. Zeroing out the weights in this row leads to zeroing the input information of the gate, making it constant. For example, zeroing the weights of the k-th row of the W_f^x and W_f^h matrices causes the forget gate of the k-th neuron to become constant, independent of the input vector $[x_t, h_{t-1}]$ and equal to $sigm(b_{f,k})$. There is no need to compute the values of constant gates after reading each input token on the forward pass through the network, therefore, constant gates can significantly speed up computation. Removing an entire hidden neuron from the model is

equivalent to zeroing out all the weights, by which the output of this neuron is multiplied when passing through the network. For the m-th hidden neuron, such weights are the m-th column of the matrix W and the m-th column of the weight matrix of the next layer. The former reflects the influence of the neuron on the hidden state of the recurrent layer h at the next time step, while the latter reflects the influence of the neuron on the neurons of the next layer (fully connected or recurrent).

To sum up, Figure 1, right, shows the selected groups of weights corresponding to individual gates and entire neurons. These groups can be sparsified both in the Bayesian framework and in the standard pruning. As a result of such sparsification: 1) some neurons are removed from the model; 2) in the remaining neurons, some of the gates become constant, which simplifies the structure of these neurons; and 3) for non-constant gates, some of the individual weights are set to zero.

Consider the implementation of the proposed structured sparsification method in standard pruning. For each neuron η, we denote the introduced (intersecting) weight groups as $w_{\eta,i}$, $w_{\eta,f}$, $w_{\eta,g}$, $w_{\eta,o}$, $w_{\eta,h}$ — the first four correspond to gates, and the last one to an entire neuron. During training, we apply an L1-regularizer on all individual weights and a group Lasso regularizer [29] to each of the introduced weight groups:

$$\lambda_1\|w\|_1 + \lambda_2 \sum_{\eta \in H}(\|w_{\eta,i}\|_2 + \|w_{\eta,f}\|_2 + \|w_{\eta,g}\|_2 + \|w_{\eta,o}\|_2 + \|w_{\eta,h}\|_2) \tag{5}$$

After the training, we set to zero all the weights with values below the threshold. As a result, because of the group Lasso, the entire weight groups are set to zero, making some gates constant and turning off some neurons.

The implementation of the proposed structured sparsification method in the Bayesian framework is based on the same principles that were described in the previous section. To sparsify individual weights, we use the Sparse variational dropout technique, which takes into account the recurrent structure of the network. To sparsify neurons, we introduce a vector of stochastic weights z^h, by which we multiply the output of the layer, similarly to [24]. To sparsify gates, we also introduce vectors of stochastic weights z^i, z^f, z^g, z^o, by which we multiply gate pre-activations. As a result, the forward pass through the LSTM recurrent layer looks as follows:

$$f_t = \sigma\Big(\big(W_f^x x_t + W_f^h h_{t-1}\big) \odot z^f + b_f\Big) \qquad \{\text{similarly for } i_t,\, o_t \text{ and } g_t\} \tag{6}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \qquad h_t = o_t \odot tanh(c_t) \odot z^h$$

Table 2: Results of structured sparsification. W — weights, G — gates, N — neurons. Compression is equal to $|w|/|w \neq 0|$. In the case of language modeling, the quality on the validation and test sets is given. The last two columns show the number of remaining neurons and non-constant gates in the recurrent layers. In the case of pruning, only the recurrent layers of the model are sparsified, while in the case of Bayesian sparsification, all layers are sparsified.

Task	Method	Quality	Compr.	Neurons	Gates
IMDb Accuracy %	Original	**84.1**	1x	128	512
	Bayes W (SparseVD-Voc)	83.62	18567	8	17
	Bayes W+N	83.98	17874x	5	12
	Bayes W+G+N	83.98	**19747x**	**4**	**6**
AGNews Accuracy %	Original	**90.6**	1x	512	2048
	Bayes W (SparseVD-Voc)	89.14	561x	34	76
	Bayes W+N	88.55	645x	17	62
	Bayes W+G+N	88.41	**647x**	**14**	**39**
Word PTB (small) Perplexity	Original	120.28 – 114.41	1x	200 – 200	800 – 800
	Pruning W+N [14]	110.34 – 106.25	1.44x	72 – 123	288 – 492
	Pruning W+G+N	**110.04 – 105.64**	**1.49x**	**64 – 115**	**193 – 442**
Word PTB (large) Perplexity	Original	82.57 – 78.57	1x	1500 – 1500	6000 – 6000
	Pruning W+N [14]	**81.25 – 77.62**	2.97x	324 – 394	1296 – 1576
	Pruning W+G+N	**81.24 – 77.82**	**3.22x**	**252 – 394**	**881 – 1418**

Zeroing the components of z^h leads to removing the corresponding neurons from the model, while zeroing the components z^i, z^f, z^g or z^o makes the corresponding gates constant and independent of the input data x_t and h_t. We work with the introduced variables z in the same way as with the rest of the weights W and the variables z for vocabulary sparsification. We use a fully-factorized log-uniform prior distribution and approximate the posterior distribution in a family of fully-factorized normal distributions.

Empirical evaluation

In Table 2 we present the results of the proposed method in the Bayesian implementation on text classification problems (data — IMDb [26] and AGNews [27]) and in the standard pruning implementation on word-level language modeling (data — Penn Treebank [25]). The proposed method improves the level of structured sparsity of the model at the level of neurons and gates without a significant quality drop. The resulting gate structures of the remaining neurons highly depend on the task (see Figure 2). For example, the last

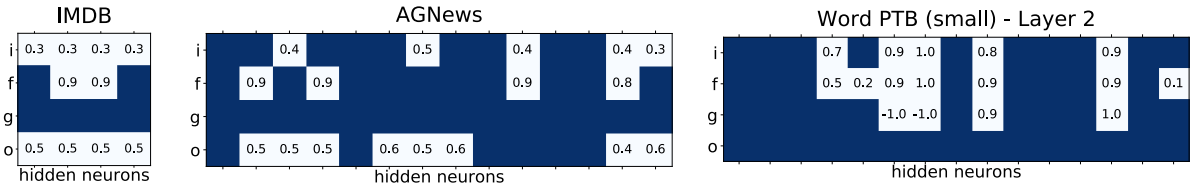

Figure 2: Gate structures for the remaining neurons obtained with the proposed structured Bayesian sparsification (Bayes W+G+N). Constant gates are shown in white with corresponding activation values. For the language modeling task, only 15 randomly chosen remaining neurons are shown.

layer of the model for the language modeling task needs to output a prediction at each time step, therefore, the output gates are important and active in all neurons. On the contrary, predictions in the classification task are made only once at the end of the input sequence, so the output gates become constant for many neurons. Moreover, we obtain similar gate structures even sparsifying only individual weights (Bayes W), although the resulting level of structured sparsity is much lower. This suggests that gate structure intrinsically exists in LSTM, and during training, neurons specialize in terms of their style of using gates. The proposed method effectively identifies and utilizes this structure to achieve better compression.

4 Neural network ensembles on a limited memory budget

The standard ensembling method for neural networks, deep ensemble [15], consists in training of n independent neural networks of the same structure from different random initializations and then averaging their predictions at the inference stage:

$$\bar{p}_{\text{obj},n} = \frac{1}{n}\sum_{i=1}^{n} p_{\text{obj},i}, \tag{7}$$

where $p_{\text{obj},i} \in [0,1]^K$ is the distribution over K classes predicted by neural network i for object obj. Ensembling several neural networks [30], as well as increasing the size of one neural network [31, 32], improves the quality of the model. Hence, if we have a limited memory budget, the question arises: on which of these two approaches should we spend the resources?

4.1 Power laws in deep ensembles

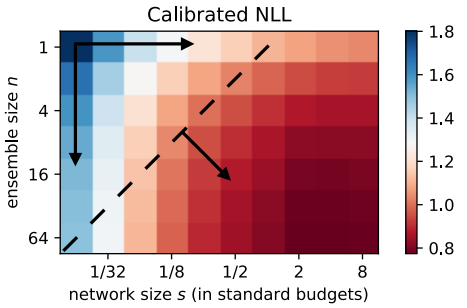

Figure 3: The behavior of ensemble CNLL with an increasing number of neural networks in the ensemble n and the size of the networks s. VGG-16 architecture on the CIFAR-100 dataset. We indicate the directions along which we study the behavior of the metrics by arrows.

Previous research shows, that a quality of an individual neural network in practice behaves as a power law with respect to the network size [16, 17]. Similar behavior has also been shown for the test error of ensembles of simple neural networks with respect to the number of networks in the ensemble [33]. In this section, we study how the test quality of an ensemble of practical convolutional neural networks behaves when we change the number of neural networks in it n, the size of these networks s, as well as the total number of parameters (see Figure 3). We name the number of networks in the ensemble as ensemble size and change the size of the networks by changing their width. As quality metrics, we consider the standard negative log-likelihood (NLL) and its calibrated version, calibrated negative log-likelihood (CNLL). The latter is more stable, better correlates with the generalization of the model, and avoids the majority of pitfalls of NLL quotepitfalls.

We show that these metrics, in many cases, behave as power laws. By power laws we mean a family of functions $\text{PL}_m = c + bm^a$, $m = 1, 2, 3, \ldots$, with parameters $a < 0$, $b \in \mathbb{R}$, $c \in \mathbb{R}$. Parameter $c = \lim_{m\to\infty} \text{PL}_m \stackrel{\text{def}}{=} \text{PL}_\infty$ reflects an asymptote to which the power law converges, and the parameters a and b specify the rate of convergence to this asymptote and the difference between the initial value and the asymptote, respectively. In experiments, we approximate the empirical ensemble quality metrics with power laws and obtain the parameters a, b, c by solving the regression problem.

Theoretical results

From a theoretical point of view, we show that NLL and the close approximation of CNLL asymptotically behave as power laws as functions of the ensemble size n. We denote a model-averaged NLL of an ensemble of n networks for a given object obj by $\text{NLL}_n^{\text{obj}} = -\mathbb{E} \log \bar{p}^*_{\text{obj},i}$. Here, the expectation is taken over all possible models that may

constitute the ensemble (e. g. random initializations). The operator $*$ denotes retrieving the element of the probability vector corresponding to the correct class of the object.

Proposition 1. *Consider an ensemble of n models, each producing independent and identically distributed probabilities of the correct class for a given object: $p^*_{\text{obj},i} \in [\epsilon_{\text{obj}}, 1]$, $\epsilon_{\text{obj}} > 0$, $i = 1, \ldots, n$. Let $\mu_{\text{obj}} = \mathbb{E}p^*_{\text{obj},i}$ and $\sigma^2_{\text{obj}} = \mathbb{D}p^*_{\text{obj},i}$ are, respectively, the mean and variance of the distribution of probabilities. Then the model-average NLL of the ensemble for a single object can be decomposed as follows:*

$$\text{NLL}_n^{\text{obj}} = \text{NLL}_\infty^{\text{obj}} + \frac{1}{n}\frac{\sigma^2_{\text{obj}}}{2\mu^2_{\text{obj}}} + \mathcal{O}\left(\frac{1}{n^2}\right). \tag{8}$$

where $\text{NLL}_\infty^{\text{obj}} = -\log(\mu_{\text{obj}})$ is the "infinite" ensemble NLL for the given object.

Summing $\text{NLL}_n^{\text{obj}}$ over objects results in the same metric for the entire dataset, NLL_n, which, by construction, also behaves as $c + bn^{-1}$ as $n \to \infty$, where $c, b > 0$ are constants with respect to n.

Similarly, we denote a model-averaged calibrated NLL of an ensemble of n networks for dataset $\mathcal{D}$:

$$\text{CNLL}_n = \mathbb{E}\min_{\tau>0}\left\{-\sum_{\text{obj}\in\mathcal{D}}\log\bar{p}^*_{\text{obj},n}(\tau)\right\}. \tag{9}$$

Calibration here implies the optimal choice of the temperature τ for the softmax function in the last layer of neural networks. When calibrating, we apply the same temperature to each neural network before the ensembling: $\bar{p}_{\text{obj},n}(\tau) = \frac{1}{n}\sum_{i=1}^n \text{softmax}\{\log(p_{\text{obj},i})/\tau\}$. If we apply the same temperature $\tau > 0$ for ensembles of all sizes n, we obtain the negative log-likelihood with a fixed temperature $\text{NLL}_n(\tau)$. This metric functionally behaves in the same way as NLL_n by definition, therefore, according to Proposition 1, it also asymptotically behaves as a power law as $n \to \infty$. Optimizing $\text{NLL}_n(\tau)$ with respect to temperature τ results in a lower envelope of the (asymptotic) power laws $\text{LE-NLL}_n = \min_{\tau>0}\text{NLL}_n(\tau)$, which also behaves as a power law as $n \to \infty$. At the same time, LE-NLL_n differs from CNLL_n only in the order of taking the expectation and minimizing with respect to τ, and in practice, these two metrics are almost identical.

Empirical results

From the theoretical analysis, we can only make conclusions about the asymptotic behavior of ensemble quality while, in practice, finite-sized ensembles are used. In the

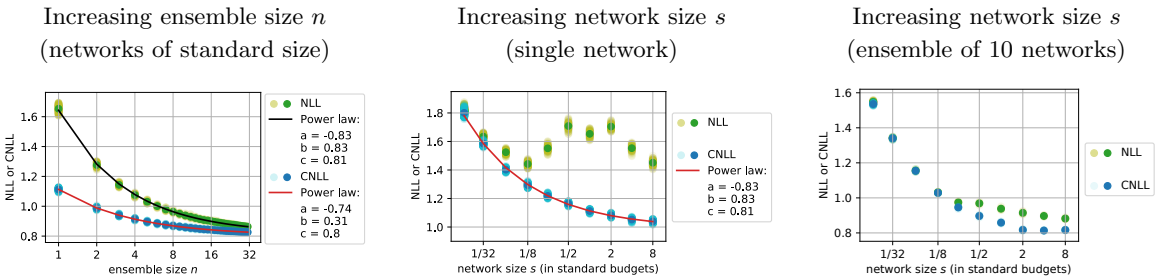

Figure 4: The behavior of NLL and CNLL of the ensemble of VGG-16 neural networks on the CIFAR-100 dataset with increasing ensemble size n and network size s.

empirical part of this section, we study how the quality of the ensemble of convolutional networks behaves in the case of finite ensemble sizes n and network sizes s. We conduct experiments with WideResNet [34] and VGG-16 [35] architectures on CIFAR-10 [36] and CIFAR-100 [37] image classification datasets. For networks of each considered size we choose the optimal hyperparameters (weight decay, learning rate and dropout rate).

The quality of the ensemble in terms of all considered metrics, $\text{NLL}_n, \text{NLL}_n(\tau)$ and CNLL_n, behaves as a power law with respect to the ensemble size n (see Figure 4, left). However, parameters a of the resulting power laws are slightly higher than the theoretical value -1 for the asymptotic behavior. An analysis of the obtained power laws allows us to draw several practically important conclusions. Firstly, the optimal temperature is not equal to 1 even for large ensembles, hence, calibration is important for large ensembles too. Secondly, large neural networks gain less from the ensembling than small ones. As a result, an ensemble of n networks of small size can be more effective than an ensemble of the same number of larger neural networks.

If we increase the size s of an individual neural network, the double descent behavior [32, 31] is observed for the NLL_s. Calibration eliminates this effect, and as a result, CNLL_s behaves as a power law with respect to the network size s. Moreover, parameter a of this power law is close to -0.5, which agrees with the results for the test error of Geiger et al. [33]. In the case of the ensemble of more than one network, increasing the network size s of an individual neural network does not lead to a power-law behavior of the quality metrics. NLL_s again experiences the double descent behavior and CNLL_s starts increasing at some network size s. The latter verifies that ensembling may be less effective for large networks than for small ones.

In addition to analyzing the behavior of the ensemble quality with respect to the number of networks n and the size of networks s separately, we study the behavior of CNLL with respect to the total number of parameters in the model. To do so, we choose the optimal ratio of n and s in terms of the resulting ensemble quality for each considered memory budget. Figure 5, left, shows that the resulting CNLL behaves as a power law when the total memory budget increases.

4.2 Memory split advantage effect

From the results on the behavior of ensemble quality with respect to the total number of parameters, we discover that for a wide range of memory budgets, the optimal ensemble consists of more than one network. We illustrate this more clearly in Figure 5, middle. For all the considered memory budgets, the optimal quality is achieved with an ensemble of at least two neural networks. This is true even for such small budgets as 1/8 of the size of one standard VGG-16 network. We observe this effect not only for CNLL but also for the classification accuracy.

We call the discovered effect a memory split advantage effect, or MSA-effect. It can be used to obtain high-quality models of small size: given a fixed memory budget, instead of training a single network, one should split the budget and train an ensemble of several smaller networks. However, the question arises: how to choose the number of networks into which a specific budget should be split? Further, we propose a method for predicting the optimal memory split based on the power laws described in the previous section.

Power laws for predictions

In the previous section, we have shown that the behavior of CNLL_n of an ensemble, with respect to the number of neural networks n, can be closely interpolated with a power law. In practice, power laws also allow extrapolating CNLL_n with high accuracy. For example, one can train ensembles of small sizes $n = 1 \dots 4$, approximate their CNLL_n with a power law, and then use this power law to predict the quality of larger ensembles of the same size networks without training them.

To predict the optimal memory split for a fixed memory budget, we propose to first predict the values for all considered splitting options, i.e. all the values on the diagonal in Figure 3 corresponding to a given budget. To do so, we train a small number of

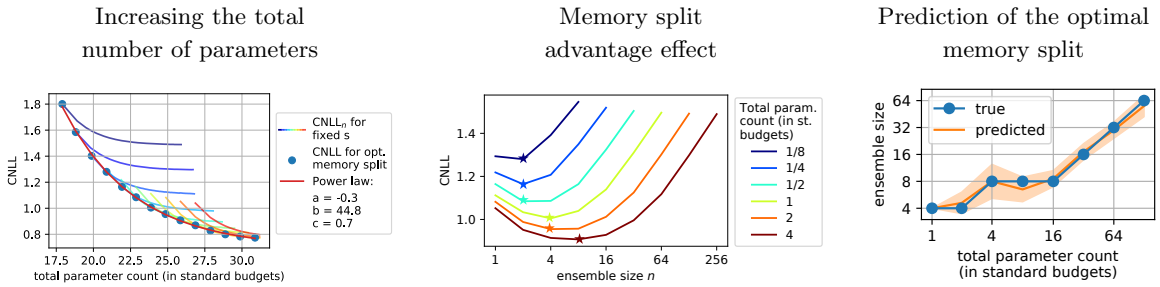

Figure 5: The behavior of CNLL of an ensemble of VGG-16 networks on the CIFAR-100 dataset when increasing the total number of parameters in the model (left), the memory split advantage effect (middle), and the prediction of optimal memory split (right). Left: the colored lines show the behavior of CNLL of an ensemble with variable ensemble size n and a fixed network size s. Center: each line corresponds to a fixed memory budget, with the optimal memory split indicated by a star. Right: predictions are shown with standard deviation obtained with 10 independent experiments.

models of each of the considered sizes, approximate the resulting CNLL_n with power laws, and then predict the diagonal values that we need. Based on the obtained diagonal values, we choose the optimal memory split and output the corresponding number of networks and their size as a prediction. Figure 5, right, shows that the described method produces accurate predictions. In practice, to reduce computations, instead of processing the networks of each considered size, we start with large ones and decrease the network size until the quality starts to deteriorate. As a result, the proposed method speeds up the choice of the optimal memory split if the optimal number of neural networks in the ensemble is $n^* \geq 4$.

5 Conclusion

In the final section, we summarize the main contributions of the work.

1. We proposed an adaptation of the Bayesian sparsification method, Sparse variational dropout [1], for recurrent neural networks. We took into account that weights in recurrent networks are used multiple times during the forward pass through the network. The obtained method allows achieving a high level of weight sparsity without a significant quality drop. We also extended this method to the sparsification of the model's vocabulary by introducing multiplicative stochastic weights, which further improves the compression results.

2. We proposed a structured sparsification method for recurrent architectures with gates based on the idea of the hierarchical sparsification of weights, gates, and neurons of the network. This method improves the level of structured sparsity of the model both when used in the Bayesian framework and in standard pruning. The analysis of the gate structures of the remaining neurons obtained using the proposed method showed that these structures are interpretable and highly depend on the task.

3. We conducted a study of the behavior of the ensemble quality of convolutional neural networks with respect to the number of networks in the ensemble and their size. We discovered that NLL and CNLL of an ensemble behave as power laws with respect to the number of networks both asymptotically and for the finite ensemble sizes in practice. We also showed that CNLL of one neural network with respect to its size and CNLL of the ensemble with respect to the total number of parameters behave as power laws. We analyzed the discovered power laws and discussed the cases when power laws were not observed.

4. We discovered the memory split advantage effect: for a fixed total number of parameters, an ensemble of several small neural networks shows better results than one large neural network. This effect appears for a wide range of memory budgets, including small budgets, such as the standard size of a single neural network used in the literature. This effect also appears both for CNLL and for the classification accuracy.

5. Based on the discovered power laws, we proposed a method for predicting the optimal memory split, i.e. the optimal number of networks in an ensemble, for a fixed memory budget. Based on a small number of trained neural networks, this method predicts the quality of larger ensembles using power-law approximations and then chooses the optimal memory split.

References

[1] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[3] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR*, arXiv:1907.11692, 2019.

[5] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A ConvNet for the 2020s. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[6] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *Deep Learning and Representation Learning Workshop at the Neural Information Processing Systems (NeurIPS)*, 2014.

[8] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with

high-dimensional output targets. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[9] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[10] Jangho Kim, Simyung Chang, and Nojun Kwak. PQK: Model compression via pruning, quantization, and knowledge distillation. In *Proceedings of the Conference of the International Speech Communication Association (INTERSPEECH)*, 2021.

[11] Zheng Li, Zijian Wang, Ming Tan, Ramesh Nallapati, Parminder Bhatia, Andrew Arnold, Bing Xiang, and Dan Roth. DQ-BART: Efficient sequence-to-sequence model via joint distillation and quantization. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8), 1997.

[13] Sharan Narang, Gregory F. Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. In *Proceedings of the International Conference for Learning Representations (ICLR)*, 2017.

[14] Wei Wen, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. Learning intrinsic sparse structures within long short-term memory. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[15] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[16] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, arXiv:2001.08361, 2020.

[17] Jonathan S Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A constructive prediction of the generalization error across scales. In *Proceedings of the International Conference for Learning Representations (ICLR)*, 2020.

[18] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. Compressing recurrent neural network with tensor train. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2017.

[19] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, arXiv:1504.00941, 2015.

[20] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2015.

[21] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference for Learning Representations (ICLR)*, 2014.

[22] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems 29 (NeurIPS)*, 2016.

[23] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian recurrent neural networks. *CoRR*, arXiv:1704.02798, 2017.

[24] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[25] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2), 1993.

[26] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL)*, 2011.

[27] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[28] Abigail See, Minh-Thang Luong, and Christopher D. Manning. Compression of neural machine translation models via pruning. In *Proceedings of the SIGNLL Conference on Computational Natural Language Learning*, 2016.

[29] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B*, 68, 2006.

[30] Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 2020.

[31] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Science*, 116(32), 2019.

[32] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[33] Mario Geiger, Arthur Jacot, Stefano Spigler, Franck Gabriel, Levent Sagun, Stéphane d'Ascoli, Giulio Biroli, Clément Hongler, and Matthieu Wyart. Scaling description of generalization with number of parameters in deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(2), 2020.

[34] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[36] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 (canadian institute for advanced research). .

[37] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-100 (canadian institute for advanced research). .