Regularization-based Pruning of Irrelevant Weights in Deep Neural Architectures

Regularization-based Pruning of Irrelevant Weights in Deep Neural Architectures

Giovanni Bonetta[1*], Matteo Ribero[1] and Rossella Cancelliere[1]

[1]Computer Science Department, University of Turin, Via Pessinetto, Turin, 10149, Italy.

*Corresponding author(s). E-mail(s): giovanni.bonetta@unito.it;
Contributing authors: matteo.ribero@edu.unito.it; rossella.cancelliere@unito.it;

Abstract

Deep neural networks exploiting million parameters are currently the norm. This is a potential issue because of the great number of computations needed for training, and the possible loss of generalization performance of overparameterized networks. We propose in this paper a method for learning sparse neural topologies via a regularization approach that identifies nonrelevant weights in any type of layer (i.e., convolutional, fully connected, attention and embedding ones) and selectively shrinks their norm while performing a standard back-propagation update for relevant layers. This technique, which is an improvement of classical weight decay, is based on the definition of a regularization term that can be added to any loss function regardless of its form, resulting in a unified general framework exploitable in many different contexts. The actual elimination of parameters identified as irrelevant is handled by an iterative pruning algorithm. To explore the possibility of an interdisciplinary use of our proposed technique, we test it on six different image classification and natural language generation tasks, among which four are based on real datasets. We reach state-of-the-art performance in one out of four imaging tasks while obtaining results better than competitors for the others and one out of two of the considered language generation tasks, both in terms of compression and metrics.

Keywords: Sparsity, Pruning, Regularization, NLP, Image Processing

1 Introduction

Deep learning models have consistently established in the past few years new state-of-the-art performances in a flood of different domains, including image processing [1–4], image captioning [5, 6], language generation [7, 8], and machine translation [9, 10].

The resources required to properly train them, however, can be prohibitive, since the number of weights used for these tasks may easily sum up to several million. These growing performance costs have therefore induced scientists to look for techniques limiting the size of neural architecture parameters.

An effective approach for reducing this complexity is sparsity, defined as the property that a substantial subset of the model weights have a value of zero (i.e., layers' weights matrices are sparse). Sparsity allows smaller computational and storage requirements, and as shown, for example, in [11] and [12], deep architectures tolerate it well.

It can shorten training time and reduce the memory footprint of regular networks to fit mobile devices, at only a small cost in accuracy. Smaller models are easier to send on edge devices and are also significantly less energy greedy, as noted in [13]: "the majority of the energy consumption comes from fetching the model parameters from the long term storage of the mobile device to its volatile memory". In addition, sparsity is also a solution for improving inference performance through overfitting control and, as suggested by [14], may lead to improved performance in transfer learning scenarios.

Two main sparsity-inducing approaches can be found in the literature, referred to as unstructured or structured, depending on whether single weights or entire structured groups of weights and/or neurons are removed.

Numerous methods have been proposed over the past few years to reach these goals: a nonexhaustive review of the recent relevant literature can be found in Section 2.

The $L2$ regularization-based techniques, detailed in Section 3, are among the most popular: they add a penalty term to the cost function to shrink the parameter values. All parameters dropping below a predefined threshold are then set to zero, thus obtaining sparse architectures.

A drawback of these methods is that neural weights' norms are all driven close to zero without accounting for weight relevance in the neural architecture, as discussed in detail in [15, 16].

Our work relies on this approach: we propose a new loss function holding a suited regularization term that is a specialization of the well-known weight decay. It allows the derivation of a new weight update rule that selectively decreases the norm of nonrelevant weights, while performing a standard backpropagation update for relevant weights. Weights' update directly follows from loss optimization, not requiring the definition of ad hoc update rules, as frequently done (see [11, 15–17]). Decreased weights are then pruned to sparsify the neural architecture.

Our technique is general, as the proposed regularization term can be added to any loss function regardless of its form and constitutes a unified framework potentially exploitable for many different applications.

We verify the effectiveness of our method in the context of image classification and natural language generation, sparsifying convolutional (LeNet-5, ResNet32/50) and self-attention transformer-based neural architectures, respectively. While the former task has already been addressed in the literature, it is still rare to find sparsity techniques applied to language generation architectures.

We reach state-of-the-art results in one out of four image classification tasks, while establishing, to the best of our knowledge, new state-of-the-art performance for the others and one out of two of the considered language generation tasks.

The rest of this paper is organized as follows: Section 2 contains an overview of related works concerning structured and unstructured pruning. In Section 3, the theoretical foundations of our model are presented, and Section 4 outlines the details of our pruning algorithm. Sections 5 and 6 describe the datasets, implementation details and results obtained in both contexts.

2 Related Works

In this section, we outline some recent approaches investigating structured and unstructured sparsity techniques. Although our paper proposes an unstructured pruning method, for the sake of completeness and for introducing some works used for comparison in Section 5, we briefly resume in the following a few recent methods for structured sparsity.

In this context, block/layer pruning, group/-filter/channel pruning, and kernel pruning approaches play an important role, particularly for convolutional-based deep networks.

Block/layer pruning ([21, 22]) aims at shrinking a network by removing entire blocks or layers; in particular, group pruning techniques are used to eliminate redundant groups. A finer-grained method is filter/channel pruning ([18, 19, 25]),which is used to eliminate redundant filters. Thus, the dimensionality of the feature maps is reduced, to the extent that even entire channels can be discarded.

Kernel pruning ([20, 23, 24])aims at removing the basic feature extraction units, the k x k matrices (kernels) within filters, generating in this way sparse networks with a fine granularity. Lin et al. [20] added a regularization term during training for pruning the connections characterized by less synaptic strength. Zhu and Pei [23] proposed progressive pruning with saliency mapping of input–output channels for solving the problem of missing channels during pruning and improving efficiency.

In the context of unstructured pruning, methods based on merely removing small norm weights are the simplest [11].

Methods grounded in Bayesian statistics constitute another possibility for achieving sparsity: the soft weight sharing (SWS) approach [12] succeeds in reducing the number of parameters to be stored, sharing the redundant parameters among layers. Weights are represented as a mixture of Gaussian distributions, where the components of the mixture are learned.

Sparse variational dropout [29] treats dropout as noise applied to the inputs of all neural layers; it represents the first attempt to use the dropout technique for sparsifying networks, paving the way for new research in the field (see, for instance, [30]). Targeted dropout [17] is another dropout-based approach that focuses on disentangling an important subset of weights from unimportant weights to obtain stable pruning. Before computing the gradients for each weight update, targeted dropout stochastically selects a set of units or weights to be dropped, using the L1 and L2 norms as a proxy for weight importance, and then computes the gradients for the remaining weights.

The authors of SNIP [31] define a saliency-based criterion on network connections to select the most important connections for the given task. Then, they prune the network accordingly in a one-shot way and finally train the model to obtain good accuracy. The drawback of this algorithm is that its one-shot nature prevents it from achieving a sparsity level comparable to iterative approaches.

Guo et al. attempted to address this issue by proposing the DNS [32] technique, where incorrectly pruned connections can be reestablished by a splicing pass if evaluated as important, making the pruning effort dynamic.

As stated in Section 1, regularization can be used as a particularly effective shrink-and-prune approach to sparsity [33]but with the drawback that all weights are indiscriminately penalized.

An effective method for avoiding this issue is presented in [15], where the idea of output-based sensitivity is introduced: weights are selectively penalized depending on their capability to induce variations in network outputs when changed. A refinement of this method is presented in [16], where state-of-the-art results are reached in image classification thanks to the introduction of loss-based sensitivity, which aims at shrinking weights that contribute the least to the final loss value.

One issue often noted with unstructured techniques is that the obtained sparse network has an irregular distribution of null weights, requiring specific software/hardware support for acceleration. For this reason, frequently no significant improvements in inference speed are observed currently, but the topic remains interesting since hardware manufacturers are developing new chips designed to exploit the unstructured pattern[1].

3 The Relevance-based Pruning Method: Theory

Regularization methods turn an original unstable, ill-posed problem into a well-posed problem; they limit the capacity of models by adding a parameter norm penalty to the loss functional L, to control overfitting and decrease the test error (see [34]).

One of the most common parameter norm penalties is L2, commonly known as Tikhonov regularization [35], ridge regression or weight decay.

In a neural context, the regularized loss $\tilde{L}$ depends on each weight $w^n_{i,j}$ belonging to layer n and connecting neurons i and j and has the form:

$$\tilde{L}(\bar{w}) = L(\bar{w}) + \lambda\|\bar{w}\|^2 = L(\bar{w}) + \lambda \sum_{n,i,j} |w^n_{i,j}|^2 \quad (1)$$

where $\bar{w}$ is the vector whose elements are $w^n_{i,j}$ and λ is the regularization parameter. The iterative application of the stochastic gradient descent algorithm (SGD) at time step t

$$w^n_{i,j}(t) \equiv w^n_{i,j}(t-1) - \eta \frac{\partial \tilde{L}(\bar{w})}{\partial w^n_{i,j}} \quad (2)$$

results in the well-known weight decay update rule

$$w^n_{i,j}(t) = w^n_{i,j}(t-1) - \eta \frac{\partial L(\bar{w})}{\partial w^n_{i,j}} - 2\lambda w^n_{i,j} \quad (3)$$

[1]https://www.graphcore.ai/

where η is the learning rate. The neural weights, therefore, are driven close to zero without taking account of their relevance in the neural architecture.

The main contribution of this work is the proposal of a new loss functional $\hat{L}$, modified with respect to eq. (1), allowing us to obtain a new selective weight decay rule that shrinks the magnitude of only those weights that are not relevant to the final error. We propose modifying the regularization term in eq. (1) by multiplying it by a coefficient that measures how much the final loss value is influenced by modifying $w^n_{i,j}$.

The quantity $|\frac{\partial L}{\partial w^n_{i,j}}|$ would seem to be a good candidate for this: small derivative values, for example, indicate that even a large variation in $w^n_{i,j}$ does not cause large loss variations, i.e., weight changes are not relevant to the final loss value. In addition, large derivative values are not interesting because they characterize relevant weights, and we aim to drive to zero (and prune) only irrelevant weights. This derivative is not upper bounded, which is a possible issue in preserving convergence properties.

Considering these requests, we therefore define the *coefficient of irrelevance* $I_{n,i,j}$ as:

$$I_{n,i,j} \equiv \exp(-|\frac{\partial L}{\partial w^n_{i,j}}|), \quad 0 < I_{n,i,j} < 1. \tag{4}$$

$I_{n,i,j}$ is bounded and assumes values near 1 for irrelevant weights and near 0 for relevant weights. Moreover, it has the useful property to be almost everywhere differentiable.

We can now define the new loss functional $\hat{L}$, modified with respect to eq. (1) to selectively limit the magnitude of weights:

$$\begin{aligned}\hat{L}(\bar{w}) &\equiv L(\bar{w}) + \lambda \sum_{n,i,j} (I_{n,i,j} \cdot |w^n_{i,j}|^2) = \\ &= L(\bar{w}) + \lambda \sum_{n,i,j} (\exp(-|\frac{\partial L}{\partial w^n_{i,j}}|) \cdot |w^n_{i,j}|^2)\end{aligned} \tag{5}$$

The iterative application of stochastic gradient descent algorithm to $\hat{L}$ allows us to derive the new weights' update rule:

$$\begin{aligned} w^n_{i,j}(t) &\equiv w^n_{i,j}(t-1) - \eta \frac{\partial \hat{\mathrm{L}}}{\partial w^n_{i,j}} = \\ &= w^n_{i,j}(t-1) - \eta \frac{\partial \mathrm{L}}{\partial w^n_{i,j}} - 2\eta\lambda \exp(-|\frac{\partial L}{\partial w^n_{i,j}}|) w^n_{i,j} \\ &- \eta\lambda |w^n_{i,j}|^2 \cdot \exp(-|\frac{\partial L}{\partial w^n_{i,j}}|)(-1) \frac{\partial}{\partial w^n_{i,j}} |\frac{\partial L}{\partial w^n_{i,j}}| = \\ &= w^n_{i,j}(t-1) - \eta \frac{\partial \mathrm{L}}{\partial w^n_{i,j}} - 2\eta\lambda \exp(-|\frac{\partial L}{\partial w^n_{i,j}}|) w^n_{i,j} \\ &+ \eta\lambda |w^n_{i,j}|^2 \cdot \exp(-|\frac{\partial L}{\partial w^n_{i,j}}|) \mathrm{sgn}(|\frac{\partial^2 L}{\partial {w^n_{i,j}}^2}|) \end{aligned} \tag{6}$$

As usually done in first-order derivative optimization methods, we can neglect the second-order derivative term, so that eq. (6) becomes:

$$w^n_{i,j}(t) = w^n_{i,j}(t-1) - \eta \frac{\partial \mathrm{L}}{\partial w^n_{i,j}} - 2\eta\lambda \exp(-|\frac{\partial L}{\partial w^n_{i,j}}|) w^n_{i,j} \tag{7}$$

Different weight updates are made in the two cases determined by the extreme values of I:

- $I \sim 0$: in this case, the weight is relevant. The third term in eq. (7) is approximately zero, so a standard update is performed, without a targeted reduction in the weight norm..
- $I \sim 1$: in this case, the weight is irrelevant. Because $I \sim 1$ implies $\frac{\partial L}{\partial w^n_{i,j}} \sim 0$, the second term in eq. (7) is approximately zero, and the update rule becomes:

$$w^n_{i,j}(t) \simeq w^n_{i,j}(t-1) - 2\eta\lambda w^n_{i,j} \tag{8}$$

We can see that in this case, the weight is actually driven to zero at each iteration, because if $w^n_{i,j} > 0$ then $\Delta w^n_{i,j} < 0$ (i.e., the norm of a positive irrelevant weight is decreased); otherwise, if $w^n_{i,j} < 0$ then $\Delta w^n_{i,j} > 0$.

For a better understanding of how the coefficient of irrelevance I behaves and affects the weight's norm, we report here as an example the histograms of initial and final (i.e., after training) distributions of these two quantities for two layers of the convolutional architecture ResNet50, which is analyzed in detail in Section 5.5.

Figure 1 shows the histogram of the irrelevance coefficient values of the weights belonging to the first convolutional layer (which contains

9,408 weights) at the beginning (blue) and end (red) of the training of this model on the ImageNet dataset. In this case, the vast majority of the weights have very low I I values for the entire training process, meaning that the corresponding weights are actually relevant and should not be pruned. Figure 2 shows the distribution of the remaining weights' norm: note that just 2,731 elements are pruned out of 9,408, which corresponds to 29% of the layer weights. This number is far below 73.85%, the percentage of pruned weights on the entire model (as reported in Table 5), and shows how relevant weights are less likely to be pruned than the other weights.

Figures 3 shows the histograms of the irrelevance coefficient values for the 18^{th} convolutional layer in the model. In this case, the majority of the I values at the beginning of the training is near 1, meaning that the corresponding weights are irrelevant and can be pruned. As seen in Figure 4, the height of the final (red) weight histogram is dramatically reduced with respect to the initial (blue) histogram because training caused a dramatic reduction in the number of weights, up to 23.0% of the initial weights. In addition, observing the final distribution of irrelevance coefficients values, we can note that it is definitely more uniform, meaning that the number of irrelevant weights is greatly decreased in percentage.

4 Pruning Algorithm Description

The pruning algorithm proceeds as follows:

1. We obtain a checkpoint from which to fine-tune. A common choice is to either find it in the literature or train it on our own. Another possibility is to use a randomly initialized checkpoint.
2. We fine-tune the checkpoint (or train it from scratch when starting from a random initialization) by using our proposed regularization term, as in Equation (5). In this step, any optimizer can be used. During fine-tuning, we evaluate the model performance on the validation set each *evaluation-interval* steps and:
   - **prune**. If the validation performance is higher than a user-defined *lower-bound*, a fixed percentage (*pruning-percentage*) of the remaining model parameters is pruned, choosing from the ones with lower magnitude.
   - **not prune**. If the validation performance is lower than the user-defined *lower-bound*, the model is not ready to be pruned, so the fine-tuning proceeds normally.
3. These last two steps of the fine-tuning process are iteratively repeated until the model reaches a validation performance plateau (so it cannot be pruned further).
4. We perform a final fine-tuning phase without regularization, aiming to obtain the best performing checkpoint.

When advanced in training, the model usually reaches an accuracy plateau, making it difficult for the algorithm to hit a pruning step. Because of this, we introduce an exponential decay schedule on λ that decreases the regularization term between the two validation steps. This procedure favors accuracy with respect to sparsification and helps the model to break the performance plateau and to cross over the lower bound, leading to further pruning.

Some hyperparameters are needed to implement this process:

- *evaluation interval*: number of steps between two validation performance assessments.
- *lower bound*: performance lower bound. It is chosen as slightly lower than the state-of-the-art performance. For the image classification tasks, the performance is measured by accuracy, while for the language generation tasks, we use the metric BLEU [36], which is briefly introduced in Section 6.
- *pruning percentage*: The percentage of remaining nonzero parameters to be pruned at every pruning step.

If not stated otherwise, hyperparameters are chosen via grid search both in the training and fine-tuning phases.

We performed all our experiments using four Nvidia TITAN RTX 24Gb GPU, and the code is available at https://github.com/giobin/Applied_Intelligence_sparsity.

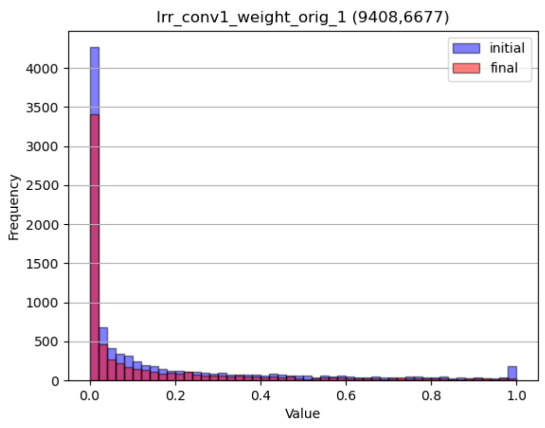

Fig. 1 Irrelevance coefficient distribution (Conv1 layer).

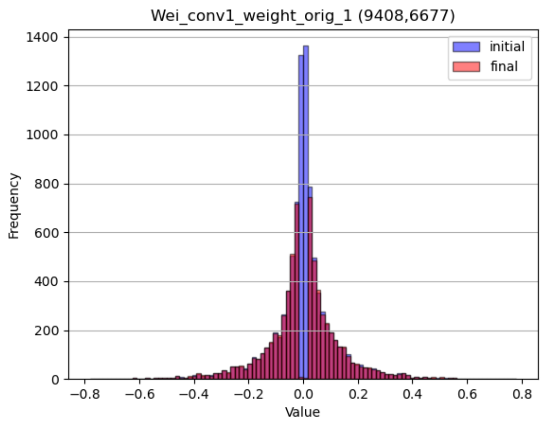

Fig. 2 Weight norm distribution (Conv1 layer). There are 6,677 remaining weights, out of 9,409.

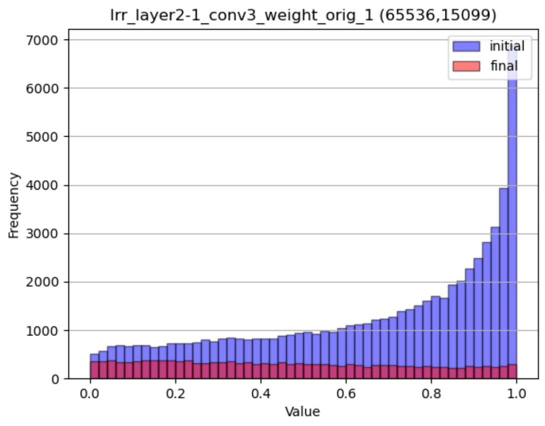

Fig. 3 Irrelevance coefficient distribution (Conv18 layer).

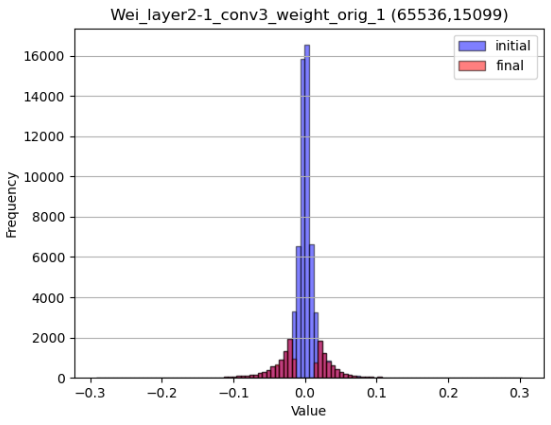

Fig. 4 Weight norm distribution (Conv18 layer). There are 15099 remaining weights, out of 65536.

5 Imaging

We test our method on four different image classification datasets chosen among the most popular benchmarks in the literature for sparsity research. Each of them is processed using architectures producing state-of-the-art performance.

5.1 Dataset Description

MNIST

MNIST [37], is composed of 70,000 28x28 grayscale images containing handwritten numbers; the dataset is split into a training set (60,000 images) and a test set (10,000 images).

Fashion-MNIST

Fashion-MNIST [38], based on the assortment of Zalando's website, is a dataset comprising 28×28 grayscale images of 70,000 fashion products from 10 categories, with 7,000 images per category. The training set has 60,000 samples, and the test set has 10,000, and, although similar to MNIST, is more challenging.

CIFAR-10

CIFAR-10 [39], from the Canadian Institute for Advanced Research, is a subset of the Tiny Images dataset [40] and consists of 60,000 32x32 color images labeled with one of 10 mutually exclusive classes: airplane, automobile, bird, cat, deer, dog,

Table 1 Hyperparameters used in imaging experiments.

Hyperparameters	MNIST/ Fashion-MNIST	CIFAR-10	ImageNet
# epochs	120	290	40
# batch size	100	128	200
η	0.001	0.0005	0.0001
Optimizer	Adam	SGD	SGD
lower-bound	98.7	92.9	75.0
λ	0.001	1e-6	1e-4
pruning-percentage	4%	4%	20%
eval-interval	250	25	500

frog, horse, ship, and truck. There are 6,000 samples per class, split into 5,000 for training and 1,000 for testing.

ImageNet

ImageNet [41] is arranged according to the WordNet [42] noun hierarchy and is a real image database, in which each node in the hierarchy is represented by thousands of images; it contains more than 20,000 categories. In total, 14 million pictures were hand-annotated, and for one million of those, the bounding boxes were also provided. The RGB images have an average size of 469x387 pixels but are usually preprocessed by sampling them at 256x256 pixels.

5.2 LeNet-5 on MNIST

Detail regarding LeNet-5 architecture are given in Table 1. This network [43] consists of a convolution (Conv) layer with 6 5x5 filters, a 2x2 pooling layer, a convolution layer with 10 5x5 filters, another 2x2 pooling layer and three fully connected (FC) layers (120, 84, 10), for a total of 431,080 parameters.

Since MNIST is an easy dataset, a pretrained checkpoint is not necessary, so we directly trained with regularization from scratch, using hyperparameter values resumed in Table 1. Finally, we fine-tuned without regularization for 5 additional epochs.

The results from our model and competitors are shown in Table 2, together with the performance of the nonsparsified baseline model. The "Sparsity(%)" column refers to the percentage of pruned weights of each model with respect to the total number of baseline weights. The "Compression Ratio" column is the ratio between the total number of weights in the baseline model and the number of remaining weights after pruning.

Performances on MNIST are very similar to each other since accuracy and sparsification on this simple task reached the top possible, i.e., very close to 100%. With our method, we obtain fewer than 1.5 k nonzero residual weights, a result primarily due to a better sparsification, when compared to the other works, of the fully connected layers, where the majority of the weights are. We obtain almost the best accuracy with the only exception of Sparse VD, even though it has higher sparsity. We also note that we obtain the same result as Han et al. but with 8% fewer weights.

Table 6 shows the disk space occupied by the pruned and unpruned models after being compressed using the GZIP[2] and BZIP2[3] algorithms with two different compression ratios, identified by -1 and -9. In particular, we can see that when using BZIP2, the pruned model is more than 20 times smaller than the unpruned model. The CPU inference time of the pruned model is $\sim$ 0.001s for one batch.

5.3 LeNet-5 on Fashion-MNIST

We obtain the initial checkpoint after training for 21 epochs. We then use the hyperparameters shown in Table 1, except for lower-bound = 90.5 and epochs = 75, for fine-tuning with regularization; finally, we fine-tune without regularization for 50 additional epochs. Table 3 compares performances on this dataset.

As can be seen, our method reaches the best performance both in terms of accuracy and compression. Our method is approximately 2 times better in compression rate than [16] with a 0.2% accuracy improvement. Similarly, our results with LeNet-5 on MNIST are due to an effective sparsification of the fully connected layers. The disk space occupied by the pruned and unpruned model after compression using GZIP and BZIP2 is reported in Table 6 and is comparable with what was obtained for the same architecture on MNIST. The CPU inference time of the pruned model is $\sim$ 0.001s for one batch.

5.4 ResNet32 on CIFAR-10

This model [3] is composed of a convolution layer with 16 3x3 filters, a batch normalization layer, 3 ResNet layers and a final dense layer, for a total of

[2]more info at:https://www.gnu.org/software/gzip/
[3]more info at: https://www.sourceware.org/bzip2/

Table 2 Test results for the LeNet-5 architecture on the MNIST dataset. (var) is the variance computed over 10 runs.

	Residual Weights (%)				Accuracy$_{(var)}$ (%)	Sparsity (%)	Compression Ratio
Methods	Conv1	Conv2	FC1	FC2			
Baseline	100	100	100	100	99.32	–	
Han et al., 2015 [11]	66	12	8	19	99.23	91.59	11.9x
Tart. et al., 2018 [15]	67.6	11.8	0.9	31.0	99.22	98.04	51.0x
DNS [32]	14	3	0.7	4	99.09	99.09	109.8x
SWS [12]	-	-	-	-	99.03	99.38	161.3x
Tart. et al., 2021 [16]	22	2.38	0.22	5.98	99.21	99.56	222.2x
L0 [44]	45	36	0.4	5	99.00	98.57	69.9x
Sparse VD [29]	33	2	0.2	5	**99.25**	99.64	277.7x
Our method	29	1.82	0.11	3.35	99.23 $_{(0.001)}$	**99.71**	**344.8x**

Table 3 Test results for LeNet-5 architecture on Fashion-MNIST dataset. (var) is the variance computed over 10 runs.

	Residual Weights (%)				Accuracy$_{(var)}$ (%)	Sparsity (%)	Compression Ratio
Methods	Conv1	Conv2	FC1	FC2			
Baseline	100	100	100	100	91.90	–	
Tart. et al., 2018 [15]	76.2	32.56	6.5	44.02	91.50	91.48	11.7x
Han et al., 2015 [11]	-	-	-	-	91.56	93.04	14.3x
Tart. et al., 2021 [16]	78.6	26.13	2.88	32.66	91.53	95.70	23.3x
Our method	78.84	17.84	1.20	6.26	**91.70** $_{(0.08)}$	**97.66**	**42.7x**

464,154 trainable parameters. All the ResNet layers are composed of 5 ResNet blocks with different configurations: they all share 2 batch normalization layers but differ in the number of kernels generated by the 2 convolutional layers (16 3x3 filters for the blocks of the first ResNet layer, 32 3x3 for the second ResNet layer and 64 3x3 for the third ResNet layer).

The initial checkpoint is obtained with the hyperparameters shown in Table 1, training for 200 epochs with $\eta = 0.1$. After fine-tuning with regularization, we fine-tune without it for 1 last epoch with a batch size = 64 and η = 6e-5.

Table 4 shows that our technique for this more challenging task, which involves a deeper neural architecture and real images, outperforms all competitors in terms of sparsity. With respect to accuracy, both our method and [16] reach the baseline values, but our method improves sparsification by 1.45% over [16]. In addition, the results show that in the case of a complex deep network with residual layers such as ResNet-32, it is possible to prune a large percentage of weights without loss in classification performance and without any

Table 4 ResNet-32 on CIFAR-10. (var) is the variance computed over 10 runs.

Methods	Accuracy$_{(var)}$ (%)	Sparsity (%)	Compression Ratio
Baseline	92.67	–	–
Sparse VD [29]	92.12	50.11	2.0x
L0 [44]	91.20	60.00	2.5x
Han et al., 2015 [11]	91.92	71.51	3.51x
Targeted Dropout [17]	92.54	80.00	5.0x
Tart. et al., 2021 [16]	**92.67**	80.11	5.0x
Our method	**92.67** $_{(0.01)}$	**81.27**	**5.33x**

change in the pruning algorithm. The disk space occupied by the pruned and unpruned model after compression using GZIP and BZIP2 is reported in Table 6; in particular we can see that using BZIP2 the pruned model is 4 times smaller on disk than the unpruned model. The CPU inference time of the pruned model is $\sim$ 0.011s for one batch.

5.5 ResNet50 on ImageNet

The model comprises a convolutional layer with batch normalization and max pooling followed by 4 ResNet layers and a final average pooling layer with a fully connected classifier on top. All ResNet layers are composed of a different number

Table 5 Test results for the ResNet50 architecture on ImageNet dataset. (var) is the variance computed over 10 runs. The upper part of the table shows the results for compression ratios up to 2.5x, while the bottom part shows the results for ratios greater than 2.5x.

Method	Baseline Accuracy (%)	Accuracy$_{(var)}$ (%)	Sparsity (%)	Compression Ratio
SSS-32, 2018 [25]	76.15	74.18	27.01	1.37x
OED, 2019 [21]	76.15	74.35	23.67	1.31x
Asympto, 2020 [26]	76.15	75.53	–	–
Hrank, 2020 [27]	76.15	74.98	36.71	1.58x
DCP, 2018 [28]	76.01	74.95	51.46	2.06x
PKPSMIO, 2022 [23]	76.09	75.86	56.15	2.28x
SSR-L2,1, 2002 [19]	76.16	73.95	37.79	1.6x
SSR-L2,0, 2020 [19]	76.16	74.00	39.36	1.64x
Our	76.16	**75.96** $_{(0.001)}$	**59.13**	**2.44x**
PKPSMIO, 2022 [23]	76.09	74.61	72.23	3.60x
Our	76.16	**74.67** $_{(0.003)}$	**73.85**	**3.82x**

Table 6 Model dimensions on disk.

Model		Gzip		bzip2	
		Gzip -1	Gzip -9	bzip2 -1	bzip2 -9
LeNet5 on MNIST (1.7MB)	Non-Pruned	1.6 MB	1.6 MB	1.7 MB	1.6 MB
	Pruned	0.24 MB	0.09 MB	0.08 MB	0.07 MB
LeNet5 on F-MNIST (1.7MB)	Non-Pruned	1.6 MB	1.6 MB	1.7 MB	1.6 MB
	Pruned	0.28 MB	0.13 MB	0.11 MB	0.10 MB
ResNet32 on CIFAR-10 (1.9MB)	Non-Pruned	1.8 MB	1.7 MB	1.8 MB	1.8 MB
	Pruned	0.62 MB	0.48 MB	0.44 MB	0.43 MB
ResNet50 on ImageNet (98MB)	Non-Pruned	91MB	91MB	95MB	93MB
	Pruned	41MB	33MB	30MB	30MB

of ResNet blocks, which comprise 3 convolutional layers interleaved with 3 batch normalization layers. Most of the convolutional layers mostly have 3×3 filters and downsampling is performed at the end of every first ResNet block relying on convolution with stride = 2. The networks contain 25,557,032 weights. For our experiments, we start the regularized finetuning from a pretrained checkpoint[4] using the hyperparameters shown in Table 1. We perform further fine-tuning for 5 epochs to obtain the best accuracy. A comparison between the results of our model and those of recent competitor works is shown in Table 5. Our technique is shown to be very effective on ResNet50, achieving up to 3.83x sparsity with respect to the baseline and with only a slight loss in accuracy ($\sim$ 1.9%). Moreover, our solution is effective in both low and high compression regimes (i.e., $<$ 2.5x and $\geq$ 2.5x), resulting in the best accuracy and sparsity levels when compared to those of all other techniques. The amount of disk space occupied by the pruned and unpruned models after compression using GZIP and BZIP2 are reported in Table 6. Notably, when using BZIP2, the pruned model requires more than 3 times less disk space than the unpruned one. The CPU inference time of the pruned model is $\sim$ 2.09s for one batch.

6 Language Generation

The first studies concerning sparsity in language architectures appeared recently [29, 33], following the progressive replacement of recurrent architectures by transformer-based models, and mainly focused on attention head pruning [45, 46].

[4]https://pytorch.org/vision/stable/index.html

Nonetheless, with respect to the large number of available sparsity techniques for imaging, it is rare to find as many results in the language generation context. We try to fill this gap by sparsifying the transformer [9] on two different language tasks: dialog learning and machine translation.

We implemented the model using the HuggingFace[5] library which provides easy access to different datasets, tokenizers and output generation techniques. Since our sparsification algorithm is not architecture specific, no modifications are needed with respect to what is described in Sec. 4.

In addition, as previously mentioned in the same section, we use the BLEU (Bilingual Evaluation Understudy) score, which compares a generated sentence to a reference sentence, as a suitable metric for evaluation in the language generation context. It works by counting matching n-grams in the candidate translation to n-grams in the reference text. A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0. BLEU was originally proposed by [36] for evaluating the predictions made by automatic machine translation systems but is now commonly used for many other language generation tasks such as dialog generation, image caption generation, text summarization and speech recognition.

6.1 Dataset Description

WMT14

WMT14 [48] is a collection of datasets presented in the Ninth Workshop on Statistical Machine Translation. It comes as a parallel corpus made by sentences translated into various languages. It is derived from many different sources, among which there are the Europarl corpus [49] (created from the European Parliament Proceedings in the official languages of the EU), the News Commentary [50] corpus and the Common Crawl corpus [51] (which was collected from web sources).

For our experiments we use the English to German translation dataset En-De WMT14, provided by the Stanford Natural Language Processing Group [52], which is more than 300x larger than Taskmaster-1; it contains 4,468,840 training samples and 3000 test samples. Some source-translation examples are shown in Table 7.

[5]https://huggingface.co/

Taskmaster-1

Taskmaster-1 [47] is a public crowdsourced dataset, released by Google in 2019, where Amazon Turkers were asked to write dyadic dialogs (see Table 8) following some given set of instructions describing six tasks: ordering pizza, creating autorepair appointments, setting up rides for hire, ordering movie tickets, ordering coffee drinks and making restaurant reservations. The dataset is composed of 13,215 task-based dialogs (12,355 for the training set and 770 for the test set), including 5,507 spoken and 7,708 written dialogs.

6.2 Transformer on WMT14

Details regarding the transformer architecture are given in Table 9 and follow the settings from [33].

To obtain the initial checkpoint, we train the model for 10 epochs with batch size = 120 and $\eta = 5e-05$, using the Adam optimizer ($\beta_1 = 0.85, \beta_2 = 0.997, eps = 1e-8$) and achieve BLEU performance comparable to the baseline defined in [33].

Starting from the checkpoint described above, the process of fine-tuning with regularization continues for 16 epochs with batch size = 100, $\eta = 2.5e-05$ and $\lambda = 2.22e-07$. Evaluations on the validation set are carried out every 6000 steps (24 times each epoch) with BLEU lower-bound = 27.3 and 10% of the remaining weights pruned when required. Finally we finetune without regularization for 5 more epochs. With respect to [33], we stop pruning when the validation performance reaches a plateau (or suddenly declines) and never surpasses the user-defined lower-bound, as described in Section 4. This criterion causes the pruning to stop at ∼80 % sparsity.

As shown in Figure 5, our pruning technique performs better than all other methods, preserving BLEU values up to ∼75% sparsity while dropping at most 0.5 points with respect to the baseline.

It also seems to be more resilient at higher compression levels since the BLEU scores start to degrade visibly only after ∼75 % sparsity is reached, whereas those of the other five pruning methods degrade earlier. This is the first experiment where our technique is used in the context of natural language generation, showing that it is very generalizable and can be effectively applied to transformer models that are heavily based on attention layers and present many shared weights,

Table 7 Example of translation pairs from En-De WMT14.

Source / **Translation**
Iron cement protects the ingot against the hot, abrasive steel casting process. Nach der Aushärtung schützt iron cement die Kokille gegen den heissen, abrasiven Stahlguss.
Goods and services advancement through the P.O.Box system is NOT ALLOWED. der Vertrieb Ihrer Waren und Dienstleistungen durch das Postfach System WIRD NICHT ZUGELASSEN.
Their achievement remains one of the greatest in recent history. Das bleibt eine der größten Errungenschaften in der jüngeren Geschichte.

Table 8 Dialog sample from Taskmaster-1.

Input
<user>Hi there, could you please help me with an order of Pizza?<enduser> <agent>Sure, where would you like to order you pizza from?<endagent> <user>I would like to order a pizza from Domino's.<enduser> <agent>What kind of pizza do you want to order? <endagent> <user>What are the specials they have right now? <enduser> <agent>There are family and party combos currently on offer <endagent > <user>No, I just want a large pizza <enduser> <agent>They have any large specialty pizza for 10.99 <endagent> <user>What are their specialty pizzas? <enduser>
Target
<agent>Well, there is the Extravagazza, Meatzza, Philly Cheesesteak, Hawaiian, Buffalo Chicken Ranch, and more. Would you like to hear more? <endagent>

Table 9 Transformer hyperparameters.

Hyperparameters	Taskmaster-1	WMT14
vocabulary	32k	32k
# encoder layers	2	6
# decoder layers	2	6
# attention heads	4	8
feed forward dim	256	2048
embedding dim.	256	512
# weights	10M	61M
max sequence len.	256	256
beam size	6	4
length penalty	-	0.6
η	0.0005	2.5e-5
Optimizer	Adam	Adam
batch size	150	120
lower-bound	6.03	27.3
λ	1e-5	2.22e-7
pruning-percentage	0.1	0.1
eval-interval	300	6000

such as in the word embedding layer and in the attention layer itself.

Table 10 shows in detail the layer-by-layer and global pruning percentages at the higher compression level reached. Table 11 shows the amount of disk space occupied by the pruned and unpruned models after being compressed using GZIP and BZIP2. Notably, when using BZIP2, the pruned model requires 4 times less disk space than the unpruned one. The CPU inference time of the pruned model is $\sim$ 4.05s for one batch.

6.3 Transformer on Taskmaster-1

Following [47], we use the dialog context up to the last user turn as input-data, and as a target for the subsequent assistant utterance. An example of this format is shown in Table 8.

Transformer architecture details are presented in [47], and shown in Table 9. We train it for 15 epochs using the Adam optimizer ($\beta_1 = 0.85, \beta_2 = 0.997, eps = 1e-8$) with batch size

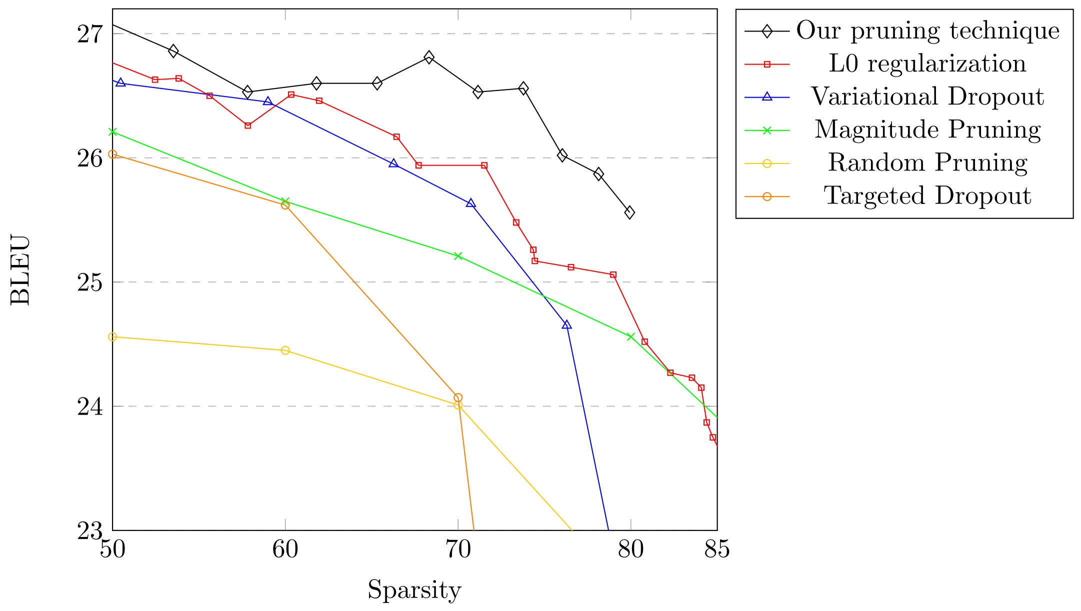

Fig. 5 BLEU results comparison at different sparsity levels on the WMT14 dataset. Except for our technique, the datapoints are taken from [17] for targeted dropout and from [33] for the others methods, for which we take only their best runs (in terms of BLEU).

Table 10 Test results for Transformer architecture on WMT14 dataset. (var) is the variance computed over 5 runs.

Model	Residual Weights (%)					BLEU$_{(var)}$	Sparsity (%)	Compression Ratio
	Encoder Attention	Decoder Attention	Encoder FFN	Decoder FFN	Embedding/ Classification Head			
Baseline	100	100	100	100	100	27.20	–	–
Our model	24.70	27.54	21.27	20.10	12.43	25.56 $_{(0.01)}$	**79.93**	**4.8x**

Table 11 Disk space dimensions of pruned/unpruned transformer models on WMT14.

Model	**GZIP**		**BZIP2**	
246.5 MB	GZIP -1	GZIP -9	BZIP2 -1	BZIP2 -9
Unpruned	228.9 MB	228.3 MB	237.1 MB	233.2 MB
Pruned	87.2 MB	66.8 MB	59.9 MB	58.2 MB

= 150 and dropout = 0.2. The final checkpoint we obtain shows comparable BLEU performance to the author's model.

Starting from this checkpoint, we fine-tune with regularization for 40 epochs with $\eta = 0.0005$. We rely on a small $\lambda = 1e - 05$ to avoid losing performance during the early stages. We find that checking every 300 training steps (i.e., 4 times every epoch) is a good compromise to obtain frequent pruning steps while retaining generation ability. The BLEU lower-bound is set to 6.03, which is very close to the author's baseline result of 6.11, and the pruning-percentage is 10%. After 30 epochs, the algorithm makes the last pruning, and the last 10 epochs are used to recover the BLEU score.

To the best of our knowledge, there are no achievements yet in the literature about weight sparsity in dialog generation tasks. We, therefore, establish the first results in this context, displayed in Table 12, testing our method against L1 and L2 regularization baselines.

Our sparsification technique allows us to obtain a highly sparsified model, with a sparsity level greater than 90%. Moreover, our final BLEU is even higher than the original result, suggesting that in some cases a sparsified model can generalize better than a nonsparsified model. In Figure 6, we show the BLEU scores of our technique and the L1 and L2 baselines at different sparsity levels. In this case, the gap between our method and the

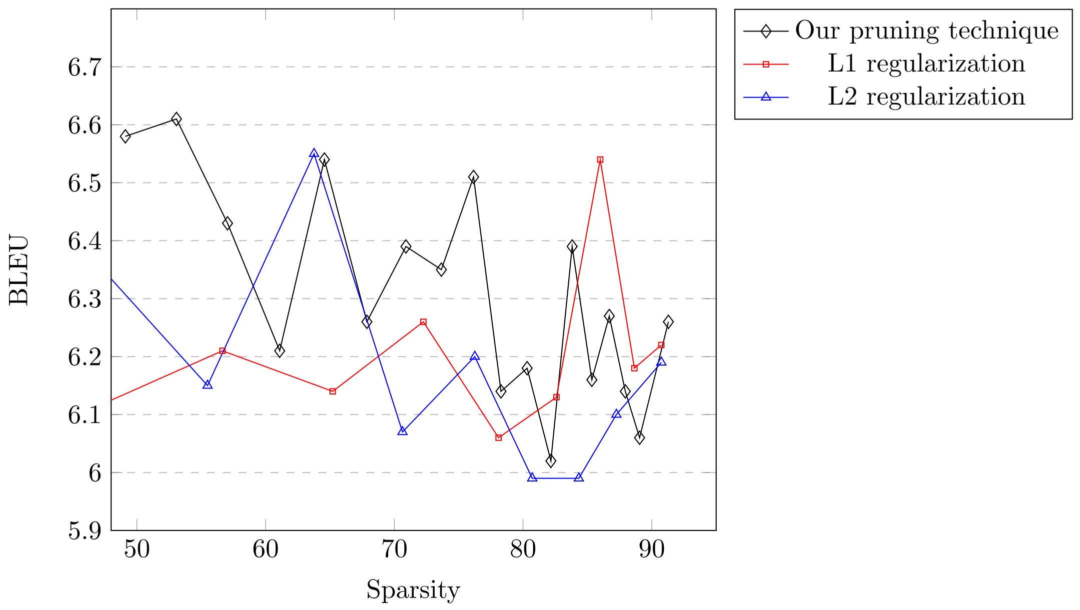

Fig. 6 BLEU results comparison at different sparsity levels on Taskmaster-1 dataset.

Table 12 Test results for Transformer architecture on Taskmaster-1 dataset. (var) is the variance computed over 10 runs.

Model	Residual Weights (%) Encoder Attention	Decoder Attention	Encoder FFN	Decoder FFN	Embedding/ Classification Head	BLEU$_{(var)}$	Sparsity (%)	Compression Ratio
Baseline	100	100	100	100	100	6.11	–	–
L1	2.53	18.57	1.17	53.08	20.99	6.22 (0.02)	90.74	10.8x
L2	17.71	21.20	15.21	26.61	6.28	6.18 (0.02)	90.74	10.8x
Our model	21.53	21.77	21.68	26.93	7.12	**6.26** (0.02)	**91.29**	11.5x

Table 13 Disk space dimensions of pruned/unpruned transformer models on Taskmaster-1.

Model	**GZIP**		**BZIP2**	
43.5 MB	GZIP -1	GZIP -9	BZIP2 -1	BZIP2 -9
Unpruned	40.2 MB	40.1 MB	41.7 MB	41.1 MB
Pruned	11.2 MB	7.4 MB	6.4 MB	6.2 MB

others is less evident due to the high variability of the BLEU scores. This is probably given by the fact that Taskmaster-1 contains rather short sentences when compared to the WMT-14 dataset, so even small output differences with the target sentence have a high impact on the final score, which is based on n-gram counting. Regardless, our system is almost always able to perform better than the L1 and L2 regularizations with the exception of sparsity levels between 0.8 and 0.9, where L1 is preferred.

Table 13 shows the disk space occupied by the pruned and unpruned models after being compressed. Additionally, in this case very good compressions can be achieved. In particular, using BZIP2, the pruned model is approximately 6,6 times smaller on the disk than the unpruned model. The CPU inference time of the pruned model is $\sim$ 2.28s for one batch.

7 Conclusions

The identification of irrelevant model parameters for pruning is the focal point of this work. We propose a solution that is an improvement of classical weight decay and consequently suitable for any functional loss. Moreover, it is simple to implement and results in a largely usable and general framework that proves to be effective in sparsifying different deep architectures.

We reach state-of-the-art results in one out of four image classification datasets and improve state-of-the-art for the others in terms of the combination of sparsity and accuracy, also obtaining a new state-of-the-art in the machine translation dataset WMT14. Since there are very few results for sparsity in language generation tasks, another contribution of this paper is that we give a new data point on Taskmaster-1.

A future interesting contribution can be to explore applications of our method to low-resource devices such as smartphones and IoT systems.

Acknowledgments. The activity has been partially carried on in the context of the Visiting Professor Program of the Gruppo Nazionale per il Calcolo Scientifico (GNCS) of the Italian Istituto Nazionale di Alta Matematica (INdAM).

References

[1] K. Zhang, L. V. Gool, R. Timofte, Deep unfolding network for image super-resolution, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[2] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, M. Li, Bag of tricks for image classification with convolutional neural networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[3] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.

[5] L. Guo, J. Liu, X. Zhu, P. Yao, S. Lu, H. Lu, Normalized and geometry-aware self-attention network for image captioning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[6] Y. Feng, L. Ma, W. Liu, J. Luo, Unsupervised image captioning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[7] R. Puduppully, L. Dong, M. Lapata, Data-to-text generation with content selection and planning, in: Proceedings of the Thirty-Third Conference on Artificial Intelligence, AAAI, Honolulu, Hawaii, USA, 2019, pp. 6908–6915.

[8] O. Dusek, J. Novikova, V. Rieser, Evaluating the state-of-the-art of end-to-end natural language generation: The E2E NLG challenge, Comput. Speech Lang. 59, 2020, pp. 123–156.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems 31, 2017, pp. 6000–6010.

[10] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: 3rd International Conference on Learning Representations ICLR, San Diego, CA, USA, 2015.

[11] S. Han, J. Pool, J. Tran, W. J. Dally, Learning both weights and connections for efficient neural network, in: C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems 28, 2015, pp. 1135–1143.

[12] K. Ullrich, E. Meeds, M. Welling, Soft weight-sharing for neural network compression, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France.

[13] V. Sanh, T. Wolf, A. M. Rush, Movement Pruning: Adaptive Sparsity by Fine-Tuning, Advances in Neural Information Processing Systems 34, 2020.

[14] Liu, J., Wang, Y., Qiao, Y., Sparse deep transfer learning for convolutional neural network, in: the Thirty-First AAAI Conference on Artificial Intelligence, AAAI 2017.

[15] E. Tartaglione, S. Lepsøy, A. Fiandrotti, G. Francini, Learning sparse neural networks via sensitivity-driven regularization, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, NeurIPS 2018.

[16] E. Tartaglione, A. Bragagnolo, A. Fiandrotti, M. Grangetto, LOss-Based SensiTivity rEgulaRization: Towards deep sparse neural networks, Neural Networks, Vol. 146, pp. 230-237, 2022.

[17] A. N. Gomez, I. Zhang, K. Swersky, Y. Gal, G. E. Hinton, Learning sparse networks using targeted dropout, arXiv preprint arXiv:1905.13678, 2019.

[18] S. Lin et al., Accelerating Convolutional Networks via Global & Dynamic Filter Pruning, Proceedings of the 27th International Joint Conference on Artificial Intelligence IJCAI, 2018.

[19] S. Lin et al., Toward Compact ConvNets via Structure-Sparsity Regularized Filter Pruning, IEEE Transactions on Neural Networks and Learning Systems, Vol. 31, N. 2, 2020, pp. 574-588.

[20] C. Lin et al., Synaptic Strength For Convolutional Neural Network, Advances in Neural Information Processing Systems 32, NeurIPS 2018.

[21] Z. Wang, S. Lin, J. Xie and Y. Lin, Pruning Blocks for CNN Compression and Acceleration via Online Ensemble Distillation, in IEEE Access, vol. 7, 2019, pp. 175703-175716.

[22] G. Ding, S. Zhang, Z. Jia, J. Zhong and J. Han, Where to Prune: Using LSTM to Guide Data-Dependent Soft Pruning, in IEEE Transactions on Image Processing, vol. 30, pp. 293-304, 2021.

[23] J. Zhu, J. Pei, Progressive kernel pruning with saliency mapping of input-output channels, Neurocomputing, Vol. 467, N. 7, 2022, pp. 360-378.

[24] J. Zhu, J. Pei, Progressive kernel pruning CNN compression method with an adjustable input channel, Applied Intelligence Vol. 52, N.3, 2022, pp. 1-22.

[25] Z. Huang, N. Wang, Data-Driven Sparse Structure Selection for Deep Neural, Proceedings of the 15th European Conference on Computer Vision ECCV, 2018.

[26] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan and Y. Yang, Asymptotic Soft Filter Pruning for Deep Convolutional Neural Networks, in IEEE Transactions on Cybernetics, vol. 50, no. 8, pp. 3594-3604, Aug. 2020.

[27] M. Lin et al., HRank: Filter Pruning Using High-Rank Feature Map, 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 1526-1535.

[28] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang and J. Zhu. Discrimination-aware channel pruning for deep neural networks. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Red Hook, NY, USA, 2018, pp. 883–894.

[29] D. Molchanov, A. Ashukha, D. P. Vetrov, Variational dropout sparsifies deep neural networks, in: D. Precup, Y. W. Teh (Eds.), Proceedings of the 34th International Conference on Machine Learning, ICML, 2017, pp. 2498–2507.

[30] H. Salehinejad, S. Valaee, EDropout: Energy-Based Dropout and Pruning of Deep Neural Networks, IEEE Transactions on Neural Networks and Learning, 2021, pp. 1-14.

[31] N. Lee, T. Ajanthan, P. H. S. Torr, Snip: single-shot network pruning based on connection sensitivity, in: Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 2019.

[32] Y. Guo, A. Yao, Y. Chen, Dynamic network surgery for efficient dnns, in: D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon,

R. Garnett (Eds.), Advances in Neural Information Processing Systems 29, Barcelona, Spain, 2016, pp. 1379–1387.

[33] T. Gale, E. Elsen, S. Hooker, The state of sparsity in deep neural networks, arXiv preprint arXiv:1902.09574, 2019.

[34] I. J. Goodfellow, Y. Bengio, A. C. Courville, Deep Learning, Adaptive computation and machine learning series, The MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2016.

[35] A. N. Tikhonov, Solution of incorrectly formulated problems and the regularization method, Soviet Math. Dokl. 4, 1963, pp. 1035–1038.

[36] K. Papineni, S. Roukos, T. Ward, W. J. Zhu, BLEU: a Method for Automatic Evaluation of Machine Translation, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 311-318.

[37] Y. LeCun, C. Cortes, MNIST handwritten digit database, 1990.

[38] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747, 2017.

[39] V. N. Alex Krizhevsky, G. Hinton, CIFAR RGB image dataset, 2009.

[40] A. Torralba, R. Fergus, W. T. Freeman, 80 million tiny images: A large data set for nonparametric object and scene recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (11), 2008, pp. 1958–1970.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: Proceedings of the 2009 IEEE conference on computer vision and pattern recognition, 2009, pp. 248–255.

[42] G. A. Miller, Wordnet: A lexical database for english, Commun. ACM 38 (11), 1995, pp 39–41.

[43] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Computation 1 (4), 1989, pp. 541–551.

[44] C. Louizos, M. Welling, D. P. Kingma, Learning sparse neural networks through L_0 regularization, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, OpenReview.net, 2018.

[45] P. Michel, O. Levy, G. Neubig, Are sixteen heads really better than one?, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 33, 2019.

[46] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, I. Titov, Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), Stroudsburg, PA, USA, 2019, pp. 5797–5808.

[47] B. Byrne, K. Krishnamoorthi, C. Sankar, A. Neelakantan, B. Goodrich, D. Duckworth, S. Yavuz, A. Dubey, K. Kim, A. Cedilnik, Taskmaster-1: Toward a realistic and diverse dialog dataset, in: K. Inui, J. Jiang, V. Ng, X. Wan (Eds.), Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing EMNLP-IJCNLP, Hong Kong, China, 2019, pp. 4515–4524.

[48] O. Bojar, C. Buck, C. Federmann, B. Haddow, P. Koehn, J. Leveling, C. Monz, P. Pecina, M. Post, H. Saint-Amand, R. Soricut, L. Specia, A. Tamchyna, in: Proceedings of the Ninth Workshop on Statistical Machine Translation, Association for Computational Linguistics, Baltimore, Maryland, USA, 2014, pp. 12–58.

[49] P. Koehn, Europarl: A Parallel Corpus for Statistical Machine Translation, in: Proceedings of the tenth Machine Translation Summit, AAMT, Phuket, Thailand, 2005, pp. 79–86.

[50] J. Tiedemann, Parallel data, tools and interfaces in opus, in: Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), European Language Resources Association (ELRA), Istanbul, Turkey, 2012.

[51] CommonCrawl, CommonCrawl's dataset, 2012.

[52] T. S. N. L. P. Group, Neural Machine Translation, 2015.