Neural Networks 143 (2021) 475–488

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Recurrent neural network pruning using dynamical systems and iterative fine-tuning

Christos Chatzikonstantinou *, Dimitrios Konstantinidis, Kosmas Dimitropoulos, Petros Daras

Information Technologies Institute, Centre for Research and Technology Hellas, Greece

ARTICLE INFO

Article history:
Received 8 February 2021
Received in revised form 24 June 2021
Accepted 2 July 2021
Available online 8 July 2021

Keywords:
Recurrent neural networks
Network pruning
Linear dynamical systems
Regularization

ABSTRACT

Network pruning techniques are widely employed to reduce the memory requirements and increase the inference speed of neural networks. This work proposes a novel RNN pruning method that considers the RNN weight matrices as collections of time-evolving signals. Such signals that represent weight vectors can be modelled using Linear Dynamical Systems (LDSs). In this way, weight vectors with similar temporal dynamics can be pruned as they have limited effect on the performance of the model. Additionally, during the fine-tuning of the pruned model, a novel discrimination-aware variation of the L2 regularization is introduced to penalize network weights (i.e., reduce the magnitude), whose impact on the output of an RNN network is minimal. Finally, an iterative fine-tuning approach is proposed that employs a bigger model to guide an increasingly smaller pruned one, as a steep decrease of the network parameters can irreversibly harm the performance of the pruned model. Extensive experimentation with different network architectures demonstrates the potential of the proposed method to create pruned models with significantly improved perplexity by at least 0.62% on the PTB dataset and improved F1-score by 1.39% on the SQuAD dataset, contrary to other state-of-the-art approaches that slightly improve or even deteriorate models' performance.

1. Introduction

Deep Neural Networks (DNNs) have revolutionized the field of machine learning due to their highly discriminative abilities and robust performance, leading to their wide adoption for providing solutions to complex problems in numerous fields, such as computer vision (LeCun et al., 2010) and language modelling (Sundermeyer et al., 2012). Nevertheless, their sensitivity to noise and missing data has led several researchers to cope with the problems of stability (Wei et al., 2021), the presence of missing data (Chen et al., 2020), non-Gaussian noise (Stojanovic et al., 2020) and various types of uncertainties, such as polytropic uncertainty (Tao et al., 2021). Furthermore, the advances in ICT technologies and the broader use of smartphones sparked a growing demand for implementing such algorithms on embedded systems. Despite their effectiveness and robustness, the use of DNNs in both powerful PCs and resource constrained embedded systems is governed by serious limitations, due to the significant demands of DNNs for computational power and energy consumption. To this end, this work focuses on the compression and acceleration of deep networks to reduce their memory requirements and increase their inference speed.

In the literature, most network compression methods target the acceleration of Convolutional Neural Networks (CNNs) (Chatzikonstantinou et al., 2020; Ding et al., 2019; Liu et al., 2019) due to their importance in image processing applications. Lately, with the advent of Industry 4.0 and the growing demand for sequence processing algorithms, techniques to accelerate Recurrent Neural Networks (RNNs) (Rumelhart et al., 1986) have gained attention. RNNs and their variants, such as Long Short-Term Memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) and Recurrent Highway Networks (RHN) (Zilly et al., 2017) are capable of processing sequences and capturing temporal dependencies between video frames or words in a text and thus they are particularly useful in action recognition and language modelling. The high accuracy of recurrent neural networks in such tasks is overshadowed by their huge requirements in run-time memory and their computational complexity that renders their usage in real-time applications inherently problematic. Moreover, the increased use of mobile devices has led several deep learning networks to be implemented in such systems of decreased computational resources. Therefore, there is a need towards the reduction of the size and complexity of

* Corresponding author.
E-mail addresses: chatziko@iti.gr (C. Chatzikonstantinou), dikonsta@iti.gr (D. Konstantinidis), dimitrop@iti.gr (K. Dimitropoulos), daras@iti.gr (P. Daras).

https://doi.org/10.1016/j.neunet.2021.07.001
0893-6080/

recurrent neural networks, while maintaining their recognition performance.

Pruning the network parameters is one of the most popular compression approaches, based on the assumption that many parameters in DNNs are often redundant and therefore they can be removed without significantly affecting the recognition performance of the network. Identifying redundant weights in a network, however, can be a challenging task since millions of mathematical operations are performed inside a network and the contributions from all weights are merged to produce the final output of the network. Existing RNN pruning methods focus on abruptly pruning entire rows or columns of the RNN weight matrices based on their magnitude (Lobacheva et al., 2020; Wen et al., 2018; Yu et al., 2019). However, such approaches do not take into consideration the position of the weights in the rows or columns of the weight matrices, which can carry important information (e.g., the dynamics of the data sequence), for the RNN performance. Moreover, magnitude-based pruning entails the risk of removing erroneously individual high magnitude weights that significantly affect the RNN output. Finally, all existing pruning methods remove weights in a single step no matter the pruning ratio, although an aggressive pruning can severely deteriorate the performance of RNNs beyond recovery.

Deviating from previous work that filter weights based on their magnitudes (Lobacheva et al., 2020; Wen et al., 2018, 2020; Yu et al., 2019), the proposed method considers the rows and columns of the RNN weight matrices as time-evolving signals. In this way, the proposed method takes into consideration not only the magnitude but also the position and the sequencing of the weights in the weight matrices, leading to a more informative selection of weights to prune and a more robust compressed representation of the weight matrices. Moreover, a novel RNN structured pruning method is proposed that iteratively prunes and fine-tunes RNN models with increasing pruning ratios. The use of an iterative pruning scheme is based on the belief that an aggressive pruning can severely degrade the performance of a network as the remaining weights cannot accurately adapt their weights to compensate for the loss of the pruned weights. On the other hand, pruning weights gradually while fine-tuning the pruned model can assist the model to retain or even improve its performance with respect to the original model. Finally, during the fine-tuning of the pruned model, a discrimination-aware variation of the traditional L_2 regularization is proposed, aiming to shrink network weights based on their contribution to the RNN output. The aim is to avoid the uniform weight shrinking achieved by the traditional L_2 regularization, allowing the pruned model to better adapt its remaining weights to compensate for the pruned ones. More specifically, the main contributions of this work are summarized as follows:

- A novel pruning method is proposed considering the rows and columns of the RNN weight matrices as time-evolving signals modelled by linear dynamical systems. The aim is to cluster signals, i.e., weight vectors, and prune the ones with similar temporal dynamics, which have limited effect on the performance of the model.
- Contrary to the traditional L_2 regularization that shrinks uniformly all weights, this work proposes a discrimination-aware variation of the L_2 regularization, whose purpose is to shrink redundant weights more strongly than significant ones during fine-tuning. This is achieved by comparing the RNN output at different time steps and introducing the difference as a weight to the L2 regularization, in order to reduce the magnitude of weights with low discriminative ability.
- A novel iterative fine-tuning scheme is proposed that allows a pruned model to achieve similar or even improved performance with respect to the original one by better adapting its weights and compensate for the loss of the pruned weights. Instead of aggressively pruning all weights at once, the proposed scheme iteratively prunes a model with increasing pruning ratio and then fine-tunes the current pruned model using the guidance of the model of the previous iteration.
- The proposed method is evaluated on a language modelling and a question answering datasets using three different recurrent models achieving state-of-the-art performance in all cases.

The remainder of the paper is organized as follows. Relevant previous work is discussed in Section Section 2, while the proposed method is presented in Section Section 3. Thorough experimental analysis and extensive comparative evaluation are provided in Section Section 4. Finally, conclusions are drawn in Section Section 5.

2. Related work

In the literature, network compression and acceleration techniques can be grouped into six main categories (Cheng et al., 2018; Gupta & Agrawal, 2020): (a) pruning methods, which aim at removing redundant network parameters to reduce computational complexity and storage requirements (Chatzikonstantinou et al., 2020; Ding et al., 2019; Wen et al., 2018, 2020); (b) quantization methods that aim to quantize network parameters (i.e, reduce the number of bits needed to store weights) (Wang et al., 2018; Xu et al., 2018; Zhang et al., 2018); (c) knowledge-distillation or teacher–student methods that are based on the notion of training a shallow student network to mimic a larger pretrained teacher network (Anil et al., 2018; Bhardwaj et al., 2019; Zagoruyko & Komodakis, 2016); (d) parameter sharing methods that reduce the model size by sharing the same weights between different weight blocks (Li et al., 2016; Ling et al., 2015; Ullrich et al., 2017); (e) low-rank approximation methods that make use of decomposition techniques to split the weight matrices into smaller ones in order to reduce the computational complexity of the network (Gusak et al., 2019; Tjandra et al., 2017; Ye et al., 2018); (f) compact network design strategies that construct low-complexity network architectures at the expense of a small classification performance reduction (e.g., replacement of fully connected layers with global average pooling operators Sandler et al., 2018, use of depthwise separable convolutions Lin et al., 2013, etc.).

Pruning the network parameters is one of the most popular compression approaches aiming at removing network parameters based on the assumption that many of them are often redundant and thereby they do not significantly affect the output of the network. Pruning methods can be further classified as structured or unstructured depending on the scheme that is followed during parameter pruning.

2.1. Unstructured pruning methods

Unstructured pruning methods prune individual network parameters without relying on the structure of weight matrices (i.e., channels, rank, neurons, etc.). As a result, unstructured methods achieve a high pruning ratio, but the pruned weight matrices are irregularly sparse leading to inefficient computations and diminishing the benefits in run-time memory and speed (Gale et al., 2020). To overcome this, special software implementations have been developed to efficiently perform sparse matrix computations (Kanellopoulos et al., 2019; Lagunas, 2020).

Early research works were focused on the development of unstructured pruning methods. Han et al. (2015) trained a network using the L_2 regularization to learn which connections are important and used this information to prune network parameters with magnitude lower than a threshold. Then fine-tuning is applied so as the remaining connections can recompense for the removed ones. In Han et al. (2017), a three-step compression framework was proposed, employing pruning, quantization and Huffman encoding. Weights are pruned based on their magnitude. Then, the remaining weights are uniformly quantized and, finally, Huffman encoding is applied to benefit from the biased distribution of the remaining weights. Furthermore, Narang et al. (2017) introduced a binary mask for each weight in the network, initially set to one. The mask was updated after each iteration using different hyperparameters for each layer. Subsequently, this mask was used to prune redundant network weights. Finally, the effect of weight initialization in magnitude-based pruned LSTM networks was studied in Yu et al. (2020), concerning the domains of natural language processing (NLP) and reinforcement learning (RL). The authors proposed the "winning ticket" initialization and showed that sparsified subnetworks initialized as winning tickets achieve a better performance compared to randomly initialized subnetworks.

2.2. Structured pruning methods

Structured pruning methods prune groups of network parameters after taking into consideration the structure of weight matrices. Structured pruning methods achieve a lower pruning ratio, which has a direct effect on the computational complexity and memory footprint of the model. Filter-pruning methods have been widely employed to structurally prune CNN filters. In particular, Ding et al. (2019) proposed a Stochastic Gradient Descent (SGD) optimization method that creates identical filters during training. All but one identical filters are pruned with minimum accuracy loss. On the other hand, You et al. (2019) multiplied the output of each filter with a trainable parameter, taking advantage of the Taylor expansion to estimate the change in the loss function. The less important filters were pruned based on the modification of the loss function. Furthermore, Chatzikonstantinou et al. (2020) proposed a magnitude based approach that prunes the filters whose weight follow the Gaussian distribution, considering their contribution to the filter output to be insignificant. Moreover, auxiliary MSE losses were used during fine-tuning to facilitate the convergence of the network. In Li et al. (2020) several subnetworks were extracted based on pre-defined constraints. Then, a scheme of adaptive batch normalization was applied to choose the best candidate in terms of accuracy.

On the other hand, vector-level pruning methods have lately gained traction especially for recurrent neural network pruning. Mao et al. (2017) studied the effect of different granularity levels (i.e., fine-grained, kernel-level, vector-level and filter-level) at the network's performance. Wen et al. (2018) used the group Lasso regularization to encourage sparsity and pruned the vectors with values lower than a threshold. Furthermore, Yu et al. (2019) handled the imbalance of the information carried by the memory cell compared to the one carried by the hidden states by zeroing out the output gates of the hidden states using the L_1 norm. In Lobacheva et al. (2020), the rows of the network weight matrices that fall below a threshold were marked as constant and thus were pruned. This method was also applied on top of the method proposed in Wen et al. (2018). Furthermore, Wen et al. (2020) introduced two binary "gate" variables that control the sparsity of each dimension of each weight matrix. Contrary to Wen et al. (2018), L_0 regularization was used to induce sparsity. Yuan et al. (2021) built a discrete space to explore different architectures of various complexities ending up to a final computationally efficient architecture that was chosen based on accuracy and sparsity objectives. Finally, Fedorov et al. (2020) employed structured pruning and quantization to generate speech enhancement recurrent models that meet specific requirements to run on microcontroller units. Vectors of weights were pruned if their magnitude was smaller than a learnable threshold. Moreover, a binary neuron was introduced that controls the update of the cell and the hidden state of LSTMs.

The method proposed in this paper can be classified as a structured vector-level pruning method. Unlike previous works that mainly rely on magnitude-based pruning, the proposed method treats each vector of the weight matrices (i.e., row or column) as a time-evolving signal with each weight in the vector playing a specific and important role for the computation of the temporal dynamics of the signal.

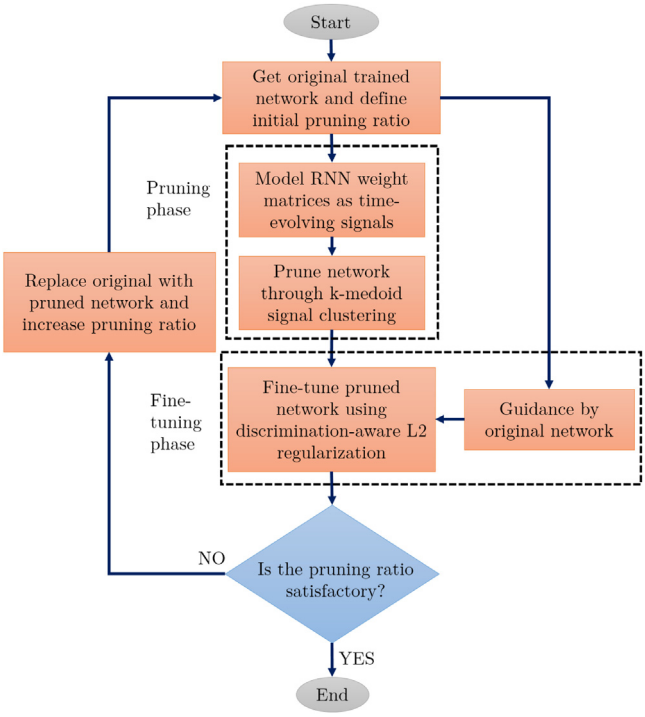

Fig. 1. [Best viewed in colour] Flowchart of the proposed RNN pruning methodology.

3. Proposed methodology

The basic steps of the proposed RNN pruning methodology are illustrated in Fig. 1, where it can be seen that the main contributions of this work are located in the pruning and fine-tuning phases, working complementary to improve the efficiency of the proposed method. More specifically, the pruning phase is responsible for the modelling, clustering and pruning of RNN weight vectors, while the fine-tuning phase is responsible for re-training the pruned model to fine-tune its weights and improve its performance. Finally, the proposed iterative scheme is a repetition of the pruning and fine-tuning phases with increasing pruning ratios until a satisfactory pruning ratio is achieved.

3.1. RNN weight matrices as collections of time-evolving signals

The pruning phase of the proposed methodology deals with the identification of the network parameters that do not contribute significantly to the output of the network. To achieve this,

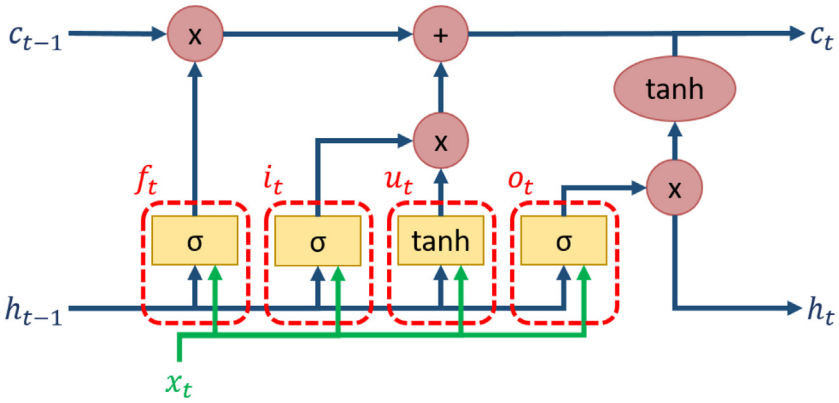

Fig. 2. [Best viewed in colour] A diagram of the operations performed inside an LSTM. The LSTM receives the input x_t at time t and the cell and hidden states c_{t-1} and h_{t-1} at time $t-1$ and outputs the cell and hidden states c_t and h_t, respectively, at time t. The forget, input and output gates, as well as the input update step are represented as f_t, i_t, o_t and u_t, respectively.

this work considers the rows and columns of the weight matrices of a recurrent neural network as discrete samples of time-evolving signals. Identifying and discarding signals with similar dynamics can significantly reduce the size and memory footprint of a recurrent neural network without affecting its performance.

Although the proposed methodology can be easily adopted to any type of RNNs, such as Gated Recurrent Units (GRUs) and Recurrent Highway Networks (RHNs), the analysis below is mainly concentrated on Long Short-Term Memory (LSTM) units, which is probably the most widely employed RNNs to date. The architecture of an LSTM, along with the operations performed inside the network, are shown in Fig. 2. Moreover, the internal input, forget, output and update weight matrices of the LSTM are shown in Eq. (1), along with their correlations with the input and hidden state vectors of the LSTM.

$$\begin{aligned} i_t &= \sigma(x_t W_{xi} + h_{t-1} W_{hi} + b_i) \\ f_t &= \sigma(x_t W_{xf} + h_{t-1} W_{hf} + b_f) \\ o_t &= \sigma(x_t W_{xo} + h_{t-1} W_{ho} + b_o) \\ u_t &= tanh(x_t W_{xu} + h_{t-1} W_{hu} + b_u) \end{aligned} \tag{1}$$

An LSTM unit consists of three different gates, the input gate (i_t), the forget gate (f_t) and the output gate (o_t) and the cell update (u_t) (Eq. (1)). The forget gate decides which part of the cell state is important and which can be ignored. The input gate decides which values of the cell state will be updated and the cell update creates the vector which will be added to the cell state. Finally, the output gate defines the parts of the cell state that will be part of the next hidden state.

From Eq. (1), it can be observed that the sizes of the weight matrices are correlated with each other in order for the LSTM equations to hold. More specifically, removing a single row from a weight matrix of the input vector W_{xk}, where $k \in \{i, f, o, u\}$ requires the reduction of the rows of all weight matrices of the input vector by one. The same holds for the weight matrices of the hidden state vector W_{hk}, where $k \in \{i, f, o, u\}$. On the other hand, removing a single column from a weight matrix of the input or hidden state vector should be followed by the reduction of the columns of the corresponding weight matrix of the hidden state or input vector by one, respectively. In other words, if a column from the weight matrix of the input vector W_{xf} is removed, then a column from the weight matrix of the hidden state vector W_{hf} needs to be discarded as well. These correlations among the weights of a weight matrix and among the weight matrices of the input and hidden state vectors are visualized in Fig. 3.

3.2. Network pruning

Developing an optimal weight pruning technique is crucial as the network size reduction should not be accompanied by a large performance degradation. To this end, the proposed network pruning method considers the modelling of rows and columns of RNN weight matrices as time-evolving signals. The aim is to discard similar sequences of weights, i.e., columns or rows, as they do not offer any novel knowledge to the RNN network. In other words, these sequences of weights do not affect significantly the performance of the network. To model the sequences of weights, a linear dynamical system (LDS) (Doretto et al., 2003; Ravichandran et al., 2012), which can be considered as a first order ARMA process with white zero mean independent and identically distributed Gaussian input, is employed. LDSs have been successfully employed in several video classification applications that require the modelling of time series (Dimitropoulos et al., 2014, 2016a, 2016b), but this is the first time they are leveraged in RNNs for the task of modelling the temporal dynamics of the sequences of weight parameters that comprise the RNN weight matrices.

The number of weight matrices can vary based on the type of RNNs, with LSTM units consisting of 8 matrices, while RHN and GRU units consisting of 6 matrices. To model the weight matrices as a collection of time-evolving signals, one can either consider each matrix individually (Fig. 3(a)) or stack them together in 3D blocks. The motivation behind the stacking of matrices in blocks lies in the relationship among the weight matrices, as described in the previous section. In the case of LSTM units, one can either consider two blocks of sizes $N \times H \times 4$ and $H \times H \times 4$ (Fig. 3(b)) by stacking the weight matrices of the input and hidden state vectors, respectively, or consider a single block of size $H \times H \times 8$ (Fig. 3(c)) by stacking all weight matrices together given that the size of the input vector N equals the number of LSTM hidden units H. In this work, stacking all matrices together is proposed due to the inherent interactions among the weight matrices. However, all weight matrix stacking options are experimentally evaluated in Section 4.3.1.

From the previous analysis and based on the selected weight matrix stacking, the block of the LSTM weight matrices can be described as follows:

$$W \in \mathbb{R}^{N \times H \times D}, D \in \{1, 4, 8\} \tag{2}$$

After having defined the structure of matrix W, the goal is to reduce the number of weight parameters and increase the training and inference speed of LSTMs, as shown in Fig. 4. More specifically, initially each row $W_{ri} \in \mathbb{R}^{H \times D}$ of the block W (plane if $D > 1$) with $i \in [0, N]$ is treated as a signal with a temporal

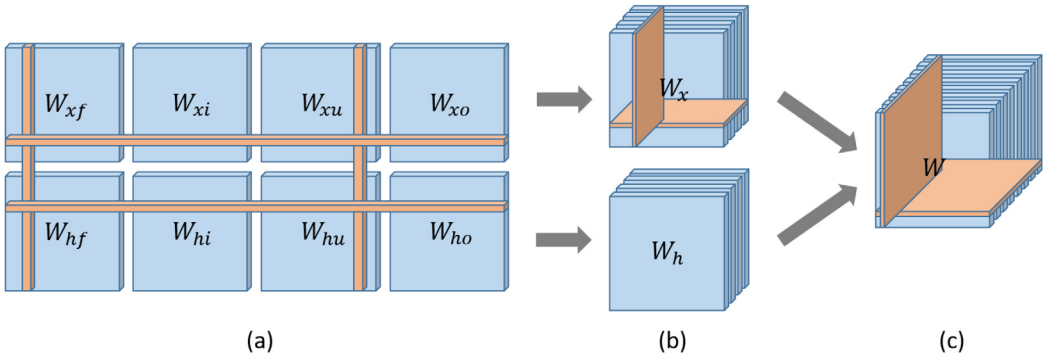

Fig. 3. [Best viewed in colour] Correlations among the shapes of the LSTM weight matrices. The removal of rows or columns from a single weight matrix should be followed by the removal of rows or columns from other weight matrices as well. The rows and columns in case of single matrices are transformed to planes when the matrices are stacked together.

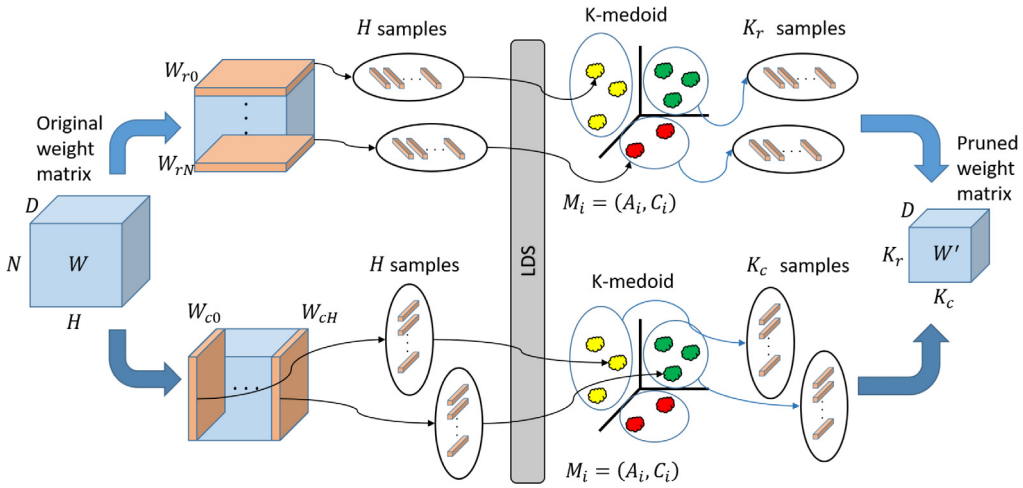

Fig. 4. [Best viewed in colour] The proposed technique for the pruning of RNN weight matrices from $N \times H$ to $K_r \times K_c$.

dimension of H and feature space D. This means that each column $w_{rc} \in \mathbb{R}^{D\times 1}$ of W_{ri} is considered as a temporal instance of the signal that evolve in time. In a similar fashion, each column $W_{cj} \in \mathbb{R}^{N\times D}$ of the block W with $j \in [0, H]$ is treated as a signal with a temporal dimension of N and feature space D. From now on, the indices i and j from the row W_{ri} and column W_{cj} of the block of weight matrices W are dropped for a clearer presentation. For each row or column of the block W, an ARMA model is formulated and modelled with a LDS using the equations below:

$$z(t+1) = Az(t) + Bv(t) \tag{3}$$

$$w_{rc} = \bar{w}_{rc} + Cz(t) + s(t) \tag{4}$$

$$\bar{w}_{rc} = \begin{cases} \frac{1}{H}\Sigma_{k=1}^{H} W_r(k), & \text{if row of } W \\ \frac{1}{N}\Sigma_{k=1}^{N} W_c(k), & \text{if column of } W \end{cases} \tag{5}$$

In Eqs. (3)–(5), $z(t) \in \mathbb{R}^{D\times 1}$ refers to the hidden state of the LDS at time t, while the quantities $s(t)$ and $Bv(t)$ are the measurement and process noise, respectively and they are considered to be equal to zero in this case. The quantities used to describe the LDS are the matrix $A \in \mathbb{R}^{D\times D}$ that models the dynamics of the hidden state of the LDS and the matrix $C \in \mathbb{R}^{D\times D}$ that maps the hidden state to the output of the system.

In this way, every row or column of the block of weight matrices W can be described by a pair of matrices $M = (A, C)$, which is a descriptor that incorporates information on the dynamics and the values of the signal incorporated in the row or column of W, respectively. To estimate the parameters of the LDS descriptor (i.e., pair of matrices M) a principal component based approach was proposed by Doretto et al. (2003). Based on this approach, a matrix Y is formed as shown below:

$$Y = \begin{cases} [W_r(1) - \bar{w}_{rc}, W_r(2) - \bar{w}_{rc}, \ldots, W_r(H) - \bar{w}_{rc}], & \text{if row of } W \\ [W_c(1) - \bar{w}_{rc}, W_c(2) - \bar{w}_{rc}, \ldots, W_c(N) - \bar{w}_{rc}], & \text{if column of } W \end{cases} \tag{6}$$

The average quantity $\bar{w}_{rc}$ is defined as in Eq. (4). Then, a singular value decomposition of the matrix Y is performed as $Y = USV^T$. The matrix C is easily computed as $C = U$, while, given that $Z = [z(1), z(2), \ldots, z(M)] = SV^T$ and $M = H$ or $M = N$ for a row or column of the block of weight matrices W, respectively, the matrix A is computed using least squares as:

$$A = \begin{cases} [z(2), z(3), \ldots, z(H)][z(1), z(2), \ldots, z(H-1)]^{+}, & \text{if row of } W \\ [z(2), z(3), \ldots, z(N)][z(1), z(2), \ldots, z(N-1)]^{+}, & \text{if column of } W \end{cases} \tag{7}$$

The symbol $+$ in Eq. (7) is used to define the pseudoinverse of the matrix that is applied to. After the modelling of the rows and columns of the block of weight matrices W using time-evolving signals, a way to identify signals with similar dynamics is required. To this end, a k-medoid algorithm is employed to cluster signals based on their dynamics and identify the most representative signals, which are the medoids of the clusters. In this way, the aim is to discard all signals of a cluster, except its medoid that it is assumed to hold enough information to

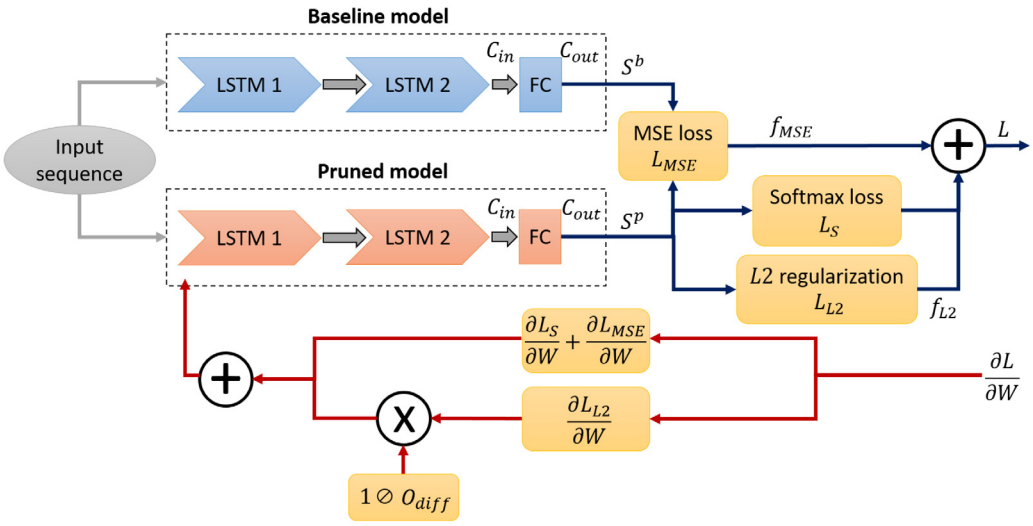

Fig. 5. Outline of the proposed network fine-tuning method with all losses, including the proposed discrimination-aware $L2$ regularization, visualized. The blue arrows depict the forward propagation, whereas the red arrows depict the backward propagation computations. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

characterize the entire cluster. The purpose of retaining just the medoids of the k-medoid clustering procedure is to significantly reduce the size and memory footprint of an RNN network without sacrificing accuracy.

To achieve this, the rows are fed to a k-medoid algorithm with predefined K_r classes. In a similar fashion, the columns are fed to another k-medoid algorithm with predefined K_c classes. The number of classes is determined by the percentage of network pruning that needs to be achieved, as the rows and columns of the weights matrices are replaced by the K_r and K_c medoids of the clusters, respectively. This means that the weight matrices of the input and hidden state vectors are pruned to the sizes of $K_r \times K_c$, while it holds that $K_r = K_c$ in the case of the weight matrices that refer to the hidden state vector. To perform k-medoid clustering, a similarity metric to compare LDS descriptors and measure their distances is required. However, the LDS descriptors do not lie in the Euclidean space as each descriptor M is defined by a pair of matrices. To overcome this problem, one family of distances that is usually employed in the literature is based on subspace angles. Given two LDS descriptors, $M_1 = (A_1, C_1)$ and $M_2 = (A_2, C_2)$, the subspace angles between them are computed by first solving for P the Lyapunov equation $A^T PA - P = -C^T C$, where

$$P = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \in \mathbb{R}^{2D \times 2D} \tag{8}$$

$$A = \begin{bmatrix} A_1 & \mathbf{0} \\ \mathbf{0} & A_2 \end{bmatrix} \in \mathbb{R}^{2D \times 2D} \tag{9}$$

$$C = \begin{bmatrix} C_1 & C_2 \end{bmatrix} \in \mathbb{R}^{D \times 2D} \tag{10}$$

After solving the above Lyapunov equation, the distance between the two LDS descriptors is defined using the Martin's distance d_M, as shown below:

$$d_M(M_1, M_2)^2 = -ln\left(\prod_{i=1}^{D} cos^2\theta_i\right), \tag{11}$$

where $cos^2\theta_i = i$th eigenvalue $\left(P_{11}^{-1} P_{12} P_{22}^{-1} P_{21}\right)$

Other similarity metrics have also been employed in the literature, such as the Geodesic distance (Chikuse, 2012) and the Grassmannian distance (Dimitropoulos et al., 2016b). These distances are based on the projection of the LDS descriptors to the Grassmann manifold. Experimental results with different distances are presented and discussed in Section 4.3.2.

3.3. Discrimination-aware L2 regularization

After the network pruning phase, the pruned network requires a fine-tuning phase in order to recover its initial performance. Auxiliary losses are often employed at this step to improve the convergence of the neural network (Chatzikonstantinou et al., 2020), as shown in Fig. 5. One such loss is the L_2 regularization that prevents the network from overfitting by shrinking the network parameters uniformly (Merity et al., 2018). This work proposes a discrimination-aware variation of the L_2 regularization that is based on the assumption that if certain rows of the RNN weight matrices produce similar output at different time steps, these rows have low discriminative ability. Consequently, these rows do not affect the network convergence and their impact to the output should be reduced more aggressively than those whose impact on the RNN output is significant. The traditional L_2 regularization term is defined as:

$$L_{L2} = f_{L2} \cdot \sum_{i=1}^{C_{in}} \cdot \sum_{j=1}^{C_{out}} (w_{ij})^2, \tag{12}$$

where w are the filters of the output layer, f_{L2} is a factor that defines the effect of the L_2 regularization to the total loss and C_{in} and C_{out} are the input and output dimensionalities of the output layer, respectively. To examine which rows have low discriminative ability, a differential vector $O_{diff} \in \mathbb{R}^{1 \times H}$, where H is equal to the RNN hidden size, is defined. Given an input sequence $\mathbf{x} = [x_1, x_2, \ldots, x_{t-1}, x_t]$ and the corresponding hidden state vectors computed from a RNN, $\mathbf{h} = [h_1, h_2, \ldots, h_{t-1}, h_t]$, each element of the vector O_{diff} is equal to the absolute difference between the consecutive RNN hidden states of the input sequence, averaged over time, as shown below:

$$O_{diff} = \sum_{rows} \begin{bmatrix} |h_2 - h_1| \\ |h_3 - h_2| \\ \cdots \\ |h_t - h_{t-1}| \end{bmatrix} / \#rows \tag{13}$$

In this way, each value of the vector O_{diff} contains the mean output difference of a single hidden state. A small mean output difference indicates a low discriminative ability of the row of the RNN weight matrices responsible for computing the corresponding hidden state. During backpropagation, the gradients of the L_{L2} regularization term are multiplied by $1 \oslash O_{diff}$, where $\oslash$ denotes

the elementwise division, as shown below:

$$(\frac{\partial L_{L2}}{\partial W_{x*}})' = 1 \oslash O_{diff} \otimes \sum_{i=1}^{N} \frac{\partial L_{L2}}{\partial W_{x*}}[:, i], \tag{14}$$

$$(\frac{\partial L_{L2}}{\partial W_{h*}})' = 1 \oslash O_{diff} \otimes \sum_{i=1}^{H} \frac{\partial L_{L2}}{\partial W_{h*}}[:, i], \tag{15}$$

$$(\frac{\partial L_{L2}}{\partial W_{b*}})' = 1 \oslash O_{diff} \otimes \frac{\partial L_{L2}}{\partial W_{b*}}, \tag{16}$$

where W_{x*} are the input matrices, W_{h*} are the hidden matrices, W_{b*} are the bias matrices, $*$ can be any of $\{f, i, u, o\}$, N is the input size, H is the hidden size of the RNN and $\otimes$ denotes the elementwise multiplication. With this operation, the rows of the RNN weight matrices producing hidden states with small mean output difference in the vector O_{diff} and thus considered insignificant for the RNN output, generate large gradients. Due to the effect of the L_2 regularization, large gradients lead the corresponding weights of the RNN weight matrices to shrink, zeroing out their effect on the output of the RNN. Experimental results presented in Section Section 4.3.5 demonstrate that the proposed discrimination-aware variation of the L_2 regularization outperforms the original one.

Moreover, a mean squared error (MSE) term is inserted after the output layer. The MSE loss can be defined as:

$$L_{MSE} = \frac{1}{C_{in}C_{out}} \cdot (S^p - S^b)^2, \tag{17}$$

where S^p denotes the output of the pruned network and S^b is the output of the baseline network. After considering all losses (shown in Fig. 5), the total loss becomes:

$$L_f = L_s + f_{MSE} * L_{MSE} + L_{L2}, \tag{18}$$

where L_s is the softmax loss and f_{MSE} is a factor that weights the impact of the MSE loss to the total loss.

3.4. Iterative network fine-tuning

In this section, a novel iterative fine-tuning approach is proposed to improve the convergence of the pruned network and thus its performance on a given task. As the performance of a network usually drops after the pruning, a fine-tuning phase is proposed in the literature to guide the pruned network to recover some of its lost accuracy or even improve the accuracy of the initial "unpruned" network that is called baseline. However, enforcing a large pruning ratio on a network may lead the network to never fully recover its initial performance and thus achieve sub-optimal classification results.

Leveraging on this, an iterative fine-tuning approach is proposed that aims to make the transition between a baseline and a highly pruned network smoother, as shown in Fig. 6. To this end, the proposed network pruning technique of Section 3.2 is employed to create n pruned models, all starting from the baseline model. Pruned model m_1 has the lowest pruning ratio, while model m_n has the highest pruning ratio. Then, the baseline model is used to fine-tune model m_1, the fine-tuned model m_1 is used to fine-tune model m_2 and the same procedure continues until the fine-tuned model m_{n-1} is employed to fine-tune model m_n. In this way, the proposed approach aims to steadily decrease the number of weights of the baseline model, while allowing the pruned model to progressively adapt its weights and compensate for the loss of the pruned weights. This approach allows the pruned model to achieve similar or even better performance with respect to the baseline model. As shown in Section 4, the final pruned network achieves a better classification performance using the proposed iterative fine-tuning procedure rather than using a single fine-tuning step.

4. Experimental evaluation

This section initially presents the datasets used for the evaluation of the proposed RNN pruning method. Then, an ablation study of the impact of different parameters on the performance of the proposed method is presented. Finally, the proposed method is compared against other state-of-the-art RNN pruning methodologies.

4.1. Datasets and metrics

Two well-known publicly available datasets on natural language processing are employed for the experimental evaluation of the proposed pruning method. These datasets are selected due to being large in size and widely employed in the literature for RNN pruning, rendering them crucial for the training and the comparative evaluation of the proposed method.

Penn Treebank (PTB): The PTB dataset (Taylor et al., 2003) consists of a vocabulary of 10k words, without capital letters, numbers, and punctuations extracted from 2,499 stories in a three-year Wall Street Journal collection. The dataset exists in both a word-level and a character-level form and in this work, the word-level form is utilized. The dataset consists of 929k training, 73k validation and 82k test words.

SQuAD: The Stanford Question Answering Dataset (SQuAD) 1.1 dataset (Rajpurkar et al., 2016) consists of over 100k questions–answers pairs that were asked by coworkers on a set of 500 Wikipedia articles. The dataset tests the ability of a system to not only answer reading comprehension questions, but also abstain when presented with a question that cannot be answered. The dataset consists of 87.6k training and 10.57k validation examples.

For the evaluation of the performance of the proposed method, the metrics of perplexity, pruning ratio and mult–add reduction are mainly employed. Perplexity measures the prediction error and it is defined as the exponential of the average loss function, i.e., e^{loss}, for all samples of the test set. A smaller perplexity indicates a better model and it is widely employed as an evaluation metric for language models. On the other hand, the pruning ratio is equal to the total number of network parameters pruned from the baseline model divided by the total number of network parameters of the baseline model. The mult–add reduction calculates the reduction in the number of arithmetic operations performed inside the model and is computed by dividing the total number of the multiplies and additions of the baseline model with the total number of the multiplies and additions of the pruned model. Finally, other important metrics for the comparison between pruned and baseline models are the model parameters, the model size that is the size of the model when saved in a hard disk, the memory size that refers to the size of run-time memory the model uses during inference and the inference time that refers to the time needed for the model to infer results for the entire test set. Regarding the comparison with other SoTA methods, the metrics chosen for the evaluation of the performance of the proposed method are the perplexity for the PTB dataset and the F1 and EM metrics for the SQuAD dataset, following the practice of previous literature works in order to ensure a fair comparison with them.

4.2. Implementation details

To train, validate and test the proposed method, the following experimental protocols were employed. The PTB dataset provides predefined data splits, while the SQuAD dataset has no test set. To this end, 10% of the SQuAD training set was kept aside for validation and the initial validation set was used as test set, a protocol commonly used by other pruning methods in the

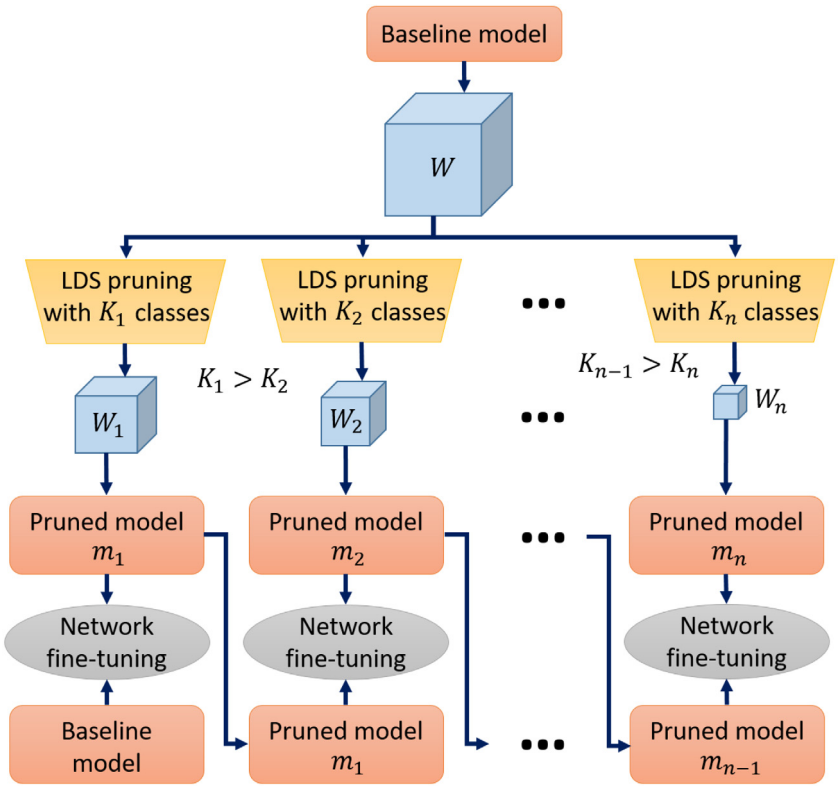

Fig. 6. [Best viewed in colour] Proposed iterative approach for the fine-tuning of an increasingly smaller model in each iteration based on the result of the previous iteration.

literature. The best model for each dataset is selected based on the metric of perplexity achieved on the test set.

Regarding the PTB dataset, the big and small PTB benchmark models are selected for pruning. Both models are composed of a word embedding layer, 2 LSTM layers and an output linear layer with a dimension of 10000 that is equal to the vocabulary size of the PTB dataset. However, the big PTB model has embedding and hidden sizes of 1500, while the small PTB model has embedding and hidden sizes of 200. Furthermore, both PTB models are evaluated with and without pruning their embedding layers. Finally, a RHN model that was initially proposed by Zilly et al. (2017) and implemented by Cruz (2019) with a width of 830 and a depth of 10 is selected for pruning and evaluation on the PTB dataset, while a variation of the Bi-Directional Attention Flow (BiDAF) model (Kim, 2018), initially proposed by Seo et al. (2017), is used for the evaluation in the SQuAD dataset. In this way, the ability of the proposed method to optimally prune any type of RNNs (i.e., LSTM, RHN and BLSTM) is stretched out.

The PTB and RHN models are trained with an initial learning rate of 20, divided by 4 each time the loss is not reduced, without using any optimizer algorithm. Gradient clipping is applied in order to avoid exploding gradients with threshold equal to 0.25. The big PTB and the RHN models are trained for 40 epochs, while the small PTB model is trained for 20 epochs. A sequence length of 35 and a batch size of 20 are used for all PTB and RHN models. The dropout value varies based on the size of the model and in particular, the baseline big PTB model is trained with a dropout of 0.65, the pruned big PTB model without its embedding layer pruned uses a dropout of 0.4, whereas the pruned big PTB model with its embedding layer pruned is trained with a dropout of 0.35. On the other hand, the small PTB model does not use dropout during training, while the RHN model is trained with a dropout of 0.3 for the hidden units and 0.65 for the output activations. The BiDAF model is trained for 12 epochs with a learning rate of 0.5 and the adadelta optimizer, a batch size of 20 and a dropout for the BLSTMs equal to 0.2. An exponential moving average algorithm is employed to smooth the weight values. Furthermore, the parameters of every model are initialized with values drawn from the uniform distribution. The values vary from -0.1 to 0.1 for the PTB models, from -0.04 to 0.04 for the RHN model and from -0.001 to 0.001 for the BiDAF model.

The proposed iterative fine-tuning approach is applied to all PTB and RHN models. However, it is not applied to the BiDAF model since its BLSTMs are not aggressively pruned. Six pruned models are created from the baseline big PTB model with hidden sizes of 1000, 700, 500, 400, 340 and 280, respectively. Two pruned models are created from the baseline small PTB model with hidden sizes of 130 and 92. Finally, three pruned models are created for the RHN model with hidden sizes of 580, 450 and 370.

For the training and testing of the implemented deep learning models, the Python 3.5 and PyTorch (Paszke et al., 2017) (version 1.4.0) environments are employed. For the computation and clustering of LDSs, the Matlab platform (version 2017a) is employed. The PC used for the experiments has an Intel Core i7-6700K with 4 GHz CPU, 32 GB RAM and a GeForce GTX 1070 GPU with 8 GB VRAM and CUDA version 10.2.

4.3. Ablation study

The ablation study showcases the importance of the novelties proposed in this work, as well as the effect of all important hyperparameters of the proposed RNN pruning method. Each novelty affects different phases of the proposed method (i.e., weight matrix modelling as temporal signals affects the pruning phase, discrimination-aware L_2 regularization affects the fine-tuning phase, while the iterative scheme affects the entire method) and thus their impact on the proposed RNN pruning

Table 1
Impact of pruning strategies on the perplexities of the big PTB and the BiDAF models.

Pruning strategy	Perplexity	
	Big PTB model (Baseline perplexity: 77.63)	BiDAF model (Baseline perplexity: 35.61)
1	84.11	31.23
4	83.71	**31.17**
8	**83.57**	n/a

Table 2
Impact of LDS similarity metrics on the perplexities of the big PTB and the BiDAF models.

LDS similarity metric	Perplexity	
	Big PTB model (Baseline perplexity: 77.63)	BiDAF model (Baseline perplexity: 35.61)
Grassmannian	83.75	31.65
Procrustes	83.78	31.97
Martin's	**83.57**	**31.17**

method is complementary. This study is performed on the test set of the PTB dataset using the big PTB model with the LSTM pruned from the size of 1500 to the size of 340 and without pruning its embedding layer and on the validation set of the SQuAD dataset using the BiDAF model. Every experiment is conducted 5 times and the mean perplexity is computed and reported.

4.3.1. Impact of pruning strategies

As it was previously discussed, for the RNN pruning, the weight matrices can be considered either individually or stacked together in 3d blocks. Due to the internal interactions among the weight matrices of an RNN, the belief is that stacking weight matrices in a single block carries the most descriptive information as it allows the modelling of their in-between dependencies. However, this work evaluates all logical combinations of weight matrix stacking to find the most beneficial one. Therefore, in the case of LSTMs of the PTB model, 3 different pruning strategies are tested: (i) Each weight matrix is considered individually (Fig. 3(a)). (ii) The weight matrices are stacked together in two blocks of 4 (Fig. 3(b)). (iii) The weight matrices are stacked together in one block of 8 (Fig. 3(c)).

In the case of BLSTMs of the BiDAF model, the third strategy is not applicable due to the fact that the input and the hidden sizes of the BLSTMs differ, i.e., the input and hidden state matrices of the BLSTMs are not equal in size and thus they cannot be stacked together, as it has been discussed in Section 3.2.

Based on the experimental results presented in Table 1, the stacking of the weight matrices in the fewer possible blocks achieves the best results, whereas the individual modelling of each weight matrix achieves the worst. This experiment validates the belief that there are correlations among the weight matrices that should be taken into consideration for the selection of an optimal pruning strategy. Considering each entry of a weight matrix as a feature in a unified space formed by the same entries of all weight matrices (i.e., feature space of 8 values in the case of LSTMs), it is possible to achieve a more accurate and robust modelling of the dynamic information present on the weight matrices, while simultaneously respecting the correlations among the features. For the remaining experiments of the ablation study, we consider the optimal choices of a single block of 8 matrices for the PTB model and two blocks of 4 matrices for the BiDAF model.

4.3.2. Impact of LDS similarity metrics

The LDS descriptors used for the modelling of the rows and columns of the RNN weight matrices do not lie in the Euclidean space and thus specialized similarity metrics are required to measure the distances among them. Three similarity metrics are widely employed in the literature, namely Martin's, Grassmannian and Procrustes distances (Lipman et al., 2011). The last two are based on the projection of the LDS descriptors to the Grassmann manifold. All three metrics are evaluated and the results are illustrated in Table 2. From the experimental results, it can be observed that the best results are achieved using the Martin's distance, while the use of the other two metrics leads to slightly deteriorated results.

4.3.3. Impact of pruning ratios

Two important hyperparameters of the k-medoid algorithm are the number of clusters used to divide the input data and the number of iterations for the convergence of the k-medoid algorithm. The k-medoid algorithm is executed with different number of iterations, ranging from 50 to 400 with a step of 50. The pruned big PTB and BiDAF models are then evaluated on the PTB and the SQuAD dataset, respectively. The experiments are inconclusive with no significant difference observed for different number of iterations. Slightly better results are achieved when the k-medoid algorithm is executed for 200 iterations and this value is adopted for all experiments.

On the other hand, the number of clusters has a direct effect on the performance of the pruned model as it significantly affects the weight pruning ratio. More specifically, Table 3 illustrates the impact of the number of clusters on the pruned model perplexity, model size, memory size, inference time, floating point operations (FLOPs) and iteration time that refers to the average time required for a single iteration of the k-medoid LDS clustering.

From the experiments, it can be observed that the model and memory size of the pruned model, as well as the iteration time of the k-medoid clustering decrease linearly with the decrease of the number of clusters whereas the inference time of the pruned model demonstrates an exponentially decreasing behaviour as the number of clusters decreases. On the other hand, the perplexity of the pruned model increases as the number of clusters decreases, showing that the model struggles to maintain its performance when the number of pruned weights increases. The selection of an appropriate number of clusters depends on the pruning ratio that needs to be achieved and the memory, speedup and performance specifications of each application.

In this work, the optimal number of clusters is determined by the pruning ratio achieved by the literature works with which the proposed method is compared. For the rest of the experiments of the ablation study, the number of clusters is equal to 340 for the PTB dataset. Regarding the SQuAD dataset, the same hidden sizes for all three BLSTMs of the BiDAF model were used for simplicity in all experiments of Table 3. In all other experiments of the ablation study and the comparative evaluation, the hidden sizes of the 3 BLSTM layers are chosen to be equal to 26, 39 and 24 respectively, as illustrated in Table 13 for fair comparison with the state-of-the-art approaches (i.e., in order to keep similar pruning ratio).

4.3.4. Impact of the MSE loss factor

This section evaluates the effect of the MSE loss factor f_{MSE} employed during network fine-tuning to the perplexity of the pruned big PTB and BiDAF models. According to the results presented in Table 4, it can be concluded that the introduction of the MSE loss term significantly improves the model performance as it decreases perplexity. As the value of the MSE loss factor increases, the performance of the model keeps improving. However, there is a threshold over which the performance of the model starts deteriorating. It is experimentally found that a value between 0.5 and 1.25, depending on the model architecture, for the MSE loss

Table 3
Number of parameters, pruning ratio, perplexity, model size, memory size, inference time, FLOPs and iteration time for the big PTB and the BiDAF models using different number of clusters.

No. of clusters	Model Parameters (M)	Pruning ratio (%)	Perplexity	Model size (MB)	Memory size (MB)	Inference time (s)	FLOPs (M)	Iteration time (s)
Big PTB model (Baseline perplexity: 77.63)								
360	22.33	66.18	83.42	89	190	0.86	44.63	119.88
340	21.84	66.92	83.57	87	185	0.77	43.65	112.84
320	21.36	67.65	84.24	86	182	0.66	42.70	106.63
300	20.89	68.36	84.35	84	179	0.63	41.76	103.83
280	20.44	69.04	84.76	82	174	0.6	40.84	97.26
260	19.99	69.73	85.45	80	170	0.58	39.94	90.85
240	19.55	70.39	86.43	78	167	0.54	39.06	84.73
BiDAF model (Baseline perplexity: 35.61)								
80	1.59	26.5	29.07	52	2805	0.240	1.74	18.83
60	1.09	49.76	32.03	50	2632	0.220	1.17	17.16
40	0.65	69.76	35.65	48	2436	0.214	0.69	15.15
20	0.29	86.50	43.81	46	2242	0.209	0.30	13.19

Table 4
Impact of the MSE loss factor on the perplexity of the big PTB and the BiDAF models.

MSE loss factor	Perplexity	
	Big PTB model (Baseline perplexity: 77.63)	BiDAF model (Baseline perplexity: 35.61)
0	83.57	31.17
0.25	77.58	28.25
0.5	**76.68**	26.98
0.75	76.82	26.33
1	76.96	25.49
1.25	77.35	**23.9**
1.5	77.77	24.67

Table 5
Impact of regularization techniques and the proposed differential vector O_{diff} on the perplexity of the big PTB and the BiDAF models.

Regularization	Perplexity	
	Big PTB model (Baseline perplexity: 77.63)	BiDAF model (Baseline perplexity: 35.61)
No regularization	76.68	23.9
L_1 regularization	76.06	23.15
Discrimination-aware L_1 regularization	76.23	22.79
L_2 regularization	76.24	22.64
Discrimination-aware L_2 regularization	**75.7**	**22.34**

Table 6
Impact of the regularization loss term factor on the perplexity of the big PTB and the BiDAF models.

Regularization loss factor	Perplexity	
	Big PTB model (Baseline perplexity: 77.63)	BiDAF model (Baseline perplexity: 35.61)
10^{-3}	94.61	24.96
10^{-4}	79.85	**22.34**
10^{-5}	**75.7**	25.11
10^{-6}	76.51	26.08
10^{-7}	76.92	26.21

factor f_{MSE} produces optimal results as far as the perplexity of the model is concerned.

4.3.5. Impact of regularization techniques

The purpose of regularization techniques is to make small modifications to the learning algorithm, enabling the model to generalize better and improve its performance. In this work, two different regularization techniques are evaluated, namely the L_1 and the L_2 regularization. Furthermore, the discrimination-aware variations of the regularization terms, as defined in Section 3.3 with the introduction of the differential vector O_{diff} are evaluated.

According to Table 5, all tested regularization techniques improve the model perplexity. Moreover, the introduction of the differential vector O_{diff}, along with the L_2 regularization further improves the results by decreasing the perplexity of the pruned big PTB model to the value of 75.7 and the perplexity of the BiDAD model to the value of 22.34. These results demonstrate that 9.96% of the total improvement in perplexity of the pruned big PTB model and 17.67% of the total improvement in perplexity of the pruned BiDAF model is attributed to the proposed discrimination-aware L_2 regularization. Thus, this experiment validates the importance of shrinking weights, whose effect to the output of a network is considered negligible. In addition, experiments are conducted, concerning the value of the factor f_{L2} that controls the impact of the regularization loss term. Based on the results of Table 6, the optimal value for the regularization loss term factor f_{L2} is equal to $1e^{-5}$ and $1e^{-4}$ for the big PTB and the BiDAF model, respectively. Higher or smaller values for the regularization loss term factor significantly deteriorate the performance of the pruned models.

4.3.6. Evaluation of the proposed iterative network fine-tuning approach

In this section, the proposed iterative fine-tuning approach is evaluated on the big PTB model. Five pruned models are created, all starting from the baseline model with hidden sizes of 1000, 700, 500, 400, and 340, respectively. Normally, the baseline model of size 1500 guides directly or in a single step the fine-tuning of the targeted pruned model, which in this case has a size of 340. However, in the proposed iterative approach, the baseline model is used to guide the model with the lowest pruning ratio and then every model guides the smaller one until the target pruning ratio is achieved.

As it is illustrated in Table 7, the proposed approach achieves a smaller perplexity of almost 2 with respect to the single-step fine-tuning approach. It is also worth noting that the pruned model using the iterative fine-tuning approach achieves a smaller perplexity even compared to the perplexity of the baseline model. This experiment validates the claim of this work that the pruning procedure should be iterative with larger and larger pruning ratios until the target is reached. Enforcing a large pruning ratio aggressively to a model can significantly deteriorate its performance beyond any recovery. Finally, Table 8 performs a direct comparison between the baseline and the pruned model in terms

Table 7
Evaluation of single-step vs iterative fine-tuning on the big PTB model.

Fine-tuning approach	Steps	Pruned model perplexity
Single-step	1500 → 340	75.7
Iterative	**1500 → 1000 → 700 → 500 → 400 → 340**	**73.73**

Table 8
Comparison of baseline and pruned big PTB models in terms of perplexity, size, memory size, inference time and FLOPs.

Model	Hidden size	Perplexity	Model Parameters (M)	Model size (MB)	Memory size (MB)	inference time (s)	FLOPs (M)
Baseline	1500	77.63	66.03	264	539	6.97	132.00
Pruned	**340**	**73.73**	**21.84**	**87**	**185**	**0.77**	43.65

Table 9
Evaluation of methods on the big PTB model without pruning its embedding layer.

Method	Baseline perplexity	Pruned perplexity	Difference	Pruning ratio (%)	Hidden size	Mult–add reduction
Wen et al. (2018)	78.57	78.65	+0.08	66.89	[373, 315]	7.48×
Lobacheva et al. (2020)	78.57	77.82	−0.75	68.94	[252, 394]	–
Yu et al. (2019)	78.57	78.37	−0.20	–	[63, 375]	8.63×
Yuan et al. (2021)	78.57	78.67	+0.10	68.3	[319, 285]	7.81×
Proposed (block size: 4)	77.63	75.86	−1.77	68.3	[319, 285]	8.7×
Proposed (block size: 8)	**77.63**	**75.25**	**−2.38**	**69.04**	**[280, 280]**	**9.41×**

Table 10
Evaluation of methods on the small PTB model without pruning its embedding layer.

Method	Baseline perplexity	Pruned perplexity	Difference	Pruning ratio (%)	Hidden size	Mult–add reduction
Lobacheva et al. (2020)	114.41	105.64	−8.77	32.89	[64, 115]	–
Proposed (block size: 4)	**113.87**	**101.42**	**−12.45**	**28.84**	**[64, 115]**	**2.03×**
Proposed (block size: 8)	**113.87**	**102.51**	**−11.36**	**33.33**	**[92, 92]**	**2.82×**

of perplexity, model size, memory size, inference time and FLOPs. The pruned model can generalize better (smaller perplexity) and run much faster due to its smaller size, storage size and run-time memory size, as well as its fewer floating operations (FLOPs). More specifically, the pruned model has three times less parameters, storage size, run-time memory size and FLOPs and nine times smaller execution time, compared to the baseline model. This conclusion verifies the ability of the proposed network pruning method to select and remove weights that are not only unnecessary but also frequently detrimental to the performance and generalization ability of the model.

4.4. Comparison with state-of-the-art methods

The proposed method is comparatively evaluated against various state-of-the-art RNN pruning methods (Lobacheva et al., 2020; Wen et al., 2018, 2020; Yu et al., 2019; Yuan et al., 2021) and the results are presented below. The compared baseline models are the same and any differences in their baseline perplexity can be attributed to the different implementation framework (i.e., PyTorch vs Tensorflow) and the stochastic nature of the training process.

Initially, a comparison is made with methods that provide results in the PTB dataset, using the big PTB model without pruning its embedding layer. Two different models are evaluated: One model employing weight matrices stacked in one block of 8 and hidden size of 280 to achieve comparable pruning ratio with other approaches and one model with two blocks of 4, employing the same hidden size with the best approach. From Table 9, it can be concluded that the best proposed model achieves a decreased perplexity by 2.38 compared to the baseline model, although with a much smaller number of parameters. Moreover, the proposed method achieves a larger decrease to the model perplexity with respect to other methods that achieve a perplexity close to the baseline model. In addition, the proposed method manages to prune a larger percentage of network parameters than the other methods. Even by using block size of 4, that is not optimal based on our ablation study, the model pruned by the proposed method achieves a larger decrease to the model perplexity compared to the other methods. Similar observations can be made by comparing the proposed method with another method that prunes the small PTB model, as shown in Table 10. These experiments demonstrate the superiority of the proposed method in identifying and pruning parameters without affecting the overall performance of the PTB models.

Contrary to the majority of the literature methods that do not prune the embedding layer, Wen et al. (2020) apply their pruning method to the embedding layer too. For fair comparison, a weight pruning method was also applied in this work by removing the weight parameters of embedding vectors starting from those with the smallest magnitude. The comparison is shown in Table 11 and reveals that the pruned model derived from the proposed method achieves a lower perplexity with a slightly larger pruning ratio than the pruned model of Wen et al. (2020). Furthermore, the proposed method achieves a larger reduction in the multiplications and the additions of the baseline model. A second model pruned by the proposed method is evaluated using the same hidden size and pruning ratio with Wen et al. (2020) that achieves an improved performance in connection with the compared method.

Moreover, the proposed method is evaluated on the PTB dataset using the RHN model. The results, shown in Table 12, demonstrate the ability of the proposed method to accurately prune parameters from various types of recurrent neural network. More specifically, the pruned model derived from the

Table 11
Evaluation of methods on the big PTB model with its embedding layer pruned.

Method	Baseline perplexity	Pruned perplexity	Difference	Pruning ratio (%)	Hidden size	Embedding size	Mult–add reduction
Wen et al. (2020)	78.57	78.08	−0.49	90.64	[296, 247]	251	13.96×
Proposed (block size: 4)	77.63	77.01	−0.62	90.64	[296, 247]	251	13.96×
Proposed (block size: 8)	**77.63**	**76.00**	**−1.63**	**90.68**	**[255, 255]**	255	**14.21×**

Table 12
Evaluation of methods that employ the RHN model on the PTB dataset.

Method	Baseline perplexity	Pruned perplexity	Difference	Pruning ratio (%)	Width	Mult–add reduction
Wen et al. (2018)	65.4	67.7	+2.3	67.66	403	–
Wen et al. (2020)	65.4	65.1	−0.3	69.36	389	–
Proposed (width 389)	**60.55**	**58.26**	**−2.29**	**69.36**	**389**	**2.87×**
Proposed (width 370)	**60.55**	**58.39**	**−2.16**	**75.13**	**370**	**3.07×**

Table 13
Hidden sizes of the forward and backward modelling layers (ModFwd and ModBwd) and the forward and backward output layer (OutFwd and OutBwd) of the BiDAF model.

Method	ModFwd1	ModBwd1	ModFwd2	ModBwd2	OutFwd1	OutBwd1	Parameters (M)
Baseline	100	100	100	100	100	100	2.16
Wen et al. (2018)	20	33	40	38	31	16	0.41
Wen et al. (2020)	26	33	36	36	33	15	0.43
Proposed	**26**	**26**	**39**	**39**	**24**	**24**	**0.40**

Table 14
Evaluation of methods on the SQuAD dataset using the BiDAF model.

Method	Baseline EM/F1	Pruned EM/F1	Difference	Pruning ratio (%)	Perplexity	Bleu	Rouge
Wen et al. (2018)	67.98/77.85	65.36/75.78	−2.62/−2.07	81.29	–	–	–
Wen et al. (2020)	67.98/77.85	65.67/75.69	−2.31/−2.16	80.36	–	–	–
Proposed	**65.28/76.08**	**63.40/74.69**	**+1.88/+1.39**	**81.30**	**22.34**	**21.67**	**45.92**

proposed method achieves the lowest perplexity and the largest improvement in perplexity with respect to the baseline model, while also achieving the largest pruning ratio, in comparison with the other pruning methods. Even lower perplexity but with a smaller pruning ratio achieves the pruned model with the same width as the model of Wen et al. (2020).

Finally, the proposed method is applied to the BiDAF model and evaluated on the SQuAD dataset. This model has four bidirectional LSTM layers, namely the contextual embedding layer, two modelling layers and the output layer. In compliance with the previous methods, the contextual embedding layer is not pruned. Table 13 presents the hidden sizes of the baseline and pruned forward and backward modelling and forward and backward output BLSTM layers and the total number of parameters of the aforementioned layers. The implementation used in this work is slightly different from the one used by Wen et al. (2018) and Wen et al. (2020) in that different word and character embedding layers are employed. Therefore, only the parameters and the parameter reduction ratio of the pruned layers are demonstrated in Tables 13 and 14 for fair comparison, along with the performance of the models measured in F1-score and Exact Match (EM) that computes the percentage of the correctly recognized sentences.

Table 14 shows that the pruned model derived from the proposed method achieves a better performance than the baseline model (increased F1-score and increased EM) with less than 19% of the number of parameters that the initial baseline model has. On the other hand, the pruned models computed from the other state-of-the-art methods demonstrate a larger drop in F1-score and EM (i.e., more than 2%) with respect to the corresponding baseline models. As a result, the proposed method can be successfully employed for the pruning of BLSTM layers as well, significantly reducing the number of parameters without deteriorating the performance of the pruned model with respect to the baseline model. The pruned model is also evaluated using additional commonly employed metrics (i.e., perplexity, bleu and rouge) in order to encourage new works on this field to evaluate their models on them.

At this point, a few advantages and limitations of the proposed method can be mentioned. The proposed method is appropriate for either small or large datasets as the time and space complexity of the method does not scale with the size of the dataset, but only with the size of the RNN weight matrices (i.e., input, hidden state and output sizes) and the predetermined number of clusters. This also means that the higher the pruning ratio, the faster the proposed network pruning method runs. On the other hand, the proposed method is not end-to-end, meaning that the model is initially trained and the RNN weight matrices are extracted using PyTorch, then the matrices are modelled using LDSs and pruned in Matlab and finally the pruned RNN matrices are loaded to the pruned model, which is fine-tuned using PyTorch. In addition, since the speed of the proposed network pruning procedure depends on the size of the RNN weight matrices, it can end up being computationally heavy for large matrices. Nevertheless, the proposed pruning procedure does not affect the execution speed of the pruned model at all.

5. Conclusion

This paper proposes a novel structured pruning method for the compression of RNN networks. Differently from other approaches that prune weights based on their magnitudes and without taking into consideration the position of the weights in the RNN matrices, the proposed method treats the rows and columns of the RNN matrices as time-evolving signals that can be modelled using LDS descriptors. In this way, signals with similar temporal dynamics can be pruned without affecting the performance of the pruned model. During fine-tuning, a discrimination-aware

variation of the L2 regularization is introduced to shrink the magnitude of weights that show limited contribution to the RNN output. Finally, a novel iterative fine-tuning scheme is proposed, in which bigger models guide increasingly smaller ones as a way to bypass the problem of performance degradation when network pruning is performed in a single step.

Directions for future work include the use of the proposed method to prune other well-known deep learning architectures and models, such as CNNs and Transformer networks, the assessment of performance gains of pruned models in real-life mobile applications and the pruning of models that process other types of sequence data, such as video and audio.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The work leading to these results received funding from the European Commission's H2020 Research Project PROTEIN under Grant Agreement No. 817732.

References

Anil, R., Pereyra, G., Passos, A., Ormandi, R., Dahl, G. E., & Hinton, G. E. (2018). Large scale distributed neural network training through online distillation. In *International conference on learning representations*.

Bhardwaj, S., Srinivasan, M., & Khapra, M. M. (2019). Efficient video classification using fewer frames. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 354–363).

Chatzikonstantinou, C., Papadopoulos, G. T., Dimitropoulos, K., & Daras, P. (2020). Neural network compression using higher-order statistics and auxiliary reconstruction losses. In *2020 IEEE/CVF conference on computer vision and pattern recognition workshops* (pp. 3077–3086).

Chen, Z., Zhang, B., Stojanovic, V., Zhang, Y., & Zhang, Z. (2020). Event-based fuzzy control for TS fuzzy networked systems with various data missing. *Neurocomputing*, *417*, 322–332.

Cheng, J., Wang, P.-s., Li, G., Hu, Q.-h., & Lu, H.-q. (2018). Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, *19*(1), 64–77.

Chikuse, Y. (2012). *Statistics on special manifolds (vol. 174)*. Springer Science & Business Media.

Cruz, L. (2019). *PyTorch language modeling*. GitHub, https://github.com/jcblaisecruz02/PyTorch-Language-Modeling.

Dimitropoulos, K., Barmpoutis, P., & Grammalidis, N. (2014). Spatio-temporal flame modeling and dynamic texture analysis for automatic video-based fire detection. *IEEE Transactions on Circuits and Systems for Video Technology*, *25*(2), 339–351.

Dimitropoulos, K., Barmpoutis, P., & Grammalidis, N. (2016). Higher order linear dynamical systems for smoke detection in video surveillance applications. *IEEE Transactions on Circuits and Systems for Video Technology*, *27*(5), 1143–1154.

Dimitropoulos, K., Barmpoutis, P., Kitsikidis, A., & Grammalidis, N. (2016). Classification of multidimensional time-evolving data using histograms of grassmannian points. *IEEE Transactions on Circuits and Systems for Video Technology*, *28*(4), 892–905.

Ding, X., Ding, G., Guo, Y., & Han, J. (2019). Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4943–4953).

Doretto, G., Chiuso, A., Wu, Y. N., & Soatto, S. (2003). Dynamic textures. *International Journal of Computer Vision*, *51*(2), 91–109.

Fedorov, I., Stamenovic, M., Jensen, C., Yang, L.-C., Mandell, A., Gan, Y., Mattina, M., & Whatmough, P. N. (2020). TinyLSTMs: Efficient neural speech enhancement for hearing aids. In *Proc. interspeech 2020* (pp. 4054–4058).

Gale, T., Zaharia, M., Young, C., & Elsen, E. (2020). Sparse GPU kernels for deep learning. In *2020 SC20: International conference for high performance computing, networking, storage and analysis* (pp. 219–232). IEEE Computer Society.

Gupta, M., & Agrawal, P. (2020). Compression of deep learning models for text: A survey. arXiv preprint arXiv:2008.05221.

Gusak, J., Kholiavchenko, M., Ponomarev, E., Markeeva, L., Blagoveschensky, P., Cichocki, A., & Oseledets, I. (2019). Automated multi-stage compression of neural networks. In *Proceedings of the IEEE international conference on computer vision workshops* (pp. 0–0).

Han, S., Mao, H., & Dally, W. J. (2017). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International conference on learning representations*.

Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems* (pp. 1135–1143).

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

Kanellopoulos, K., Vijaykumar, N., Giannoula, C., Azizi, R., Koppula, S., Ghiasi, N. M., Shahroodi, T., Luna, J. G., & Mutlu, O. (2019). Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture* (pp. 600–614).

Kim, T. (2018). *BiDAF-pytorch*. GitHub, https://github.com/galsang/BiDAF-pytorch.

Lagunas, F. (2020). *Fast block sparse matrices for pytorch*. GitHub, https://github.com/huggingface/pytorch_block_sparse.

LeCun, Y., Kavukcuoglu, K., & Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems* (pp. 253–256).

Li, X., Qin, T., Yang, J., & Liu, T.-Y. (2016). LightRNN: Memory and computation-efficient recurrent neural networks. In *Advances in neural information processing systems* (pp. 4385–4393).

Li, B., Wu, B., Su, J., & Wang, G. (2020). Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *European conference on computer vision* (pp. 639–654). Springer.

Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.

Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., & Luís, T. (2015). Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1520–1530).

Lipman, Y., Al-Aifari, R., & Daubechies, I. (2011). The continuous procrustes distance between two surfaces. arXiv preprint arXiv:1106.4588.

Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2019). Rethinking the value of network pruning. In *International conference on learning representations*.

Lobacheva, E., Chirkova, N., Markovich, A., & Vetrov, D. P. (2020). Structured sparsification of gated recurrent neural networks. In *AAAI*.

Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., & Dally, W. J. (2017). Exploring the granularity of sparsity in convolutional neural networks. In *2017 IEEE conference on computer vision and pattern recognition workshops* (pp. 1927–1934).

Merity, S., Keskar, N. S., & Socher, R. (2018). Regularizing and optimizing LSTM language models. In *International conference on learning representations*.

Narang, S., Elsen, E., Diamos, G., & Sengupta, S. (2017). Exploring sparsity in recurrent neural networks. In *International conference on learning representations*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS-W*.

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250.

Ravichandran, A., Chaudhry, R., & Vidal, R. (2012). Categorizing dynamic textures using a bag of dynamical systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(2), 342–353.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510–4520).

Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. (2017). Bidirectional attention flow for machine comprehension. In *International conference on learning representations*.

Stojanovic, V., He, S., & Zhang, B. (2020). State and parameter joint estimation of linear stochastic systems in presence of faults and non-Gaussian noises. *International Journal of Robust and Nonlinear Control*, *30*(16), 6683–6700.

Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

Tao, H., Li, X., Paszke, W., Stojanovic, V., & Yang, H. (2021). Robust PD-type iterative learning control for discrete systems with multiple time-delays subjected to polytopic uncertainty and restricted frequency-domain. *Multidimensional Systems and Signal Processing*, *32*(2), 671–692.

Taylor, A., Marcus, M., & Santorini, B. (2003). The penn treebank: An overview. In *Treebanks* (pp. 5–22). Springer.

Tjandra, A., Sakti, S., & Nakamura, S. (2017). Compressing recurrent neural network with tensor train. In *2017 international joint conference on neural networks* (pp. 4451–4458). IEEE.

Ullrich, K., Meeds, E., & Welling, M. (2017). Soft weight-sharing for neural network compression. In *International conference on learning representations*.

Wang, P., Xie, X., Deng, L., Li, G., Wang, D., & Xie, Y. (2018). Hitnet: Hybrid ternary recurrent neural network. In *Advances in neural information processing systems* (pp. 604–614).

Wei, T., Li, X., & Stojanovic, V. (2021). Input-to-state stability of impulsive reaction–diffusion neural networks with infinite distributed delays. *Nonlinear Dynamics*, *103*(2), 1733–1755.

Wen, W., He, Y., Rajbhandari, S., Zhang, M., Wang, W., Liu, F., Hu, B., Chen, Y., & Li, H. (2018). Learning intrinsic sparse structures within long short-term memory. In *International conference on learning representations*.

Wen, L., Zhang, X., Bai, H., & Xu, Z. (2020). Structured pruning of recurrent neural networks through neuron selection. *Neural Networks*, *123*, 134–141.

Xu, C., Yao, J., Lin, Z., Ou, W., Cao, Y., Wang, Z., & Zha, H. (2018). Alternating multi-bit quantization for recurrent neural networksg. In *International conference on learning representations*.

Ye, J., Wang, L., Li, G., Chen, D., Zhe, S., Chu, X., & Xu, Z. (2018). Learning compact recurrent neural networks with block-term tensor decomposition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 9378–9387).

You, Z., Yan, K., Ye, J., Ma, M., & Wang, P. (2019). Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 2130–2141).

Yu, H., Edunov, S., Tian, Y., & Morcos, A. S. (2020). Playing the lottery with rewards and multiple languages: Lottery tickets in rl and. In *International conference on learning representations*.

Yu, N., Weber, C., & Hu, X. (2019). Learning sparse hidden states in long short-term memory. In *International conference on artificial neural networks* (pp. 288–298). Springer.

Yuan, X., Savarese, P., & Maire, M. (2021). Growing efficient deep networks by structured continuous sparsification. In *International conference on learning representations*.

Zagoruyko, S., & Komodakis, N. (2016). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. arXiv preprint arXiv:1612.03928.

Zhang, D., Yang, J., Ye, D., & Hua, G. (2018). Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision* (pp. 365–382).

Zilly, J. G., Srivastava, R. K., Koutnık, J., & Schmidhuber, J. (2017). Recurrent highway networks. In *International conference on machine learning* (pp. 4189–4198). PMLR.