



Temperature-based pruning for input features in Graph Neural Networks

Lapenna Michela^{1,a} , Faglioni Francesco^{2,b}, Fioresi Rita^{3,c}, Bruno Giovanni^{4,5,d}

¹ Department of Physics and Astronomy, University of Bologna, Bologna, Italy

² Department of Chemical and Geological Sciences, University of Modena and Reggio Emilia, Modena, Italy

³ Department of Pharmacy and Biotechnologies, University of Bologna, Bologna, Italy

⁴ Federal Institute For Materials Research and Testing, BAM, Berlin, Germany

⁵ Institute of Physics and Astronomy, University of Potsdam, Potsdam, Germany

Received: 13 January 2025 / Accepted: 26 August 2025

© The Author(s) 2025

Abstract In the present work, we employ the concept of neural network temperature to prune unimportant features in input to a Graph Neural Network (GNN) architecture. In benchmark datasets for node and graph property prediction, each node comes equipped with a vector of numerous features. It is paramount to understand which information is actually necessary and which can be discarded, both for efficiency and explainability. The temperature is linked to the gradient activity due to the loss function minimization and leads to pruning of weight structures associated with small gradients. This study is done on different GNN architectures, one for node classification and another one for link prediction, and several benchmark datasets are employed. We compare the results with similar experiments previously conducted on the filters of Convolutional Neural Networks. Although still at the proof-of-concept stage, our temperature-based pruning technique stands as a promising alternative to state-of-the-art magnitude-based pruning techniques.

1 Introduction

The purpose of the present paper is to employ the notion of temperature to prune input features in Graph Neural Networks (GNN) and make a comparison with the results in [1] for the filters of Convolutional Neural Networks (CNN).

Neural networks [2] have largely been used as *black boxes*, given their outstanding ability to learn from data, despite the lack of a clear relationship between the input and the output representation. The problem of *interpretability* and *explainability* of deep learning algorithms has now become an important issue in order to understand and guide an effective and efficient usage of this technology [3, 4].

One way to interpret neural networks is through the lens of statistical physics. Since the pioneering works by Jaynes [5] and Hopfield [6], up to the theory of (deep) Boltzmann machines [7, 8], the language of statistical mechanics has proven successful in describing inference and learning. Nowadays, many researchers are trying to exploit the connections between statistical mechanics and artificial neural networks to theoretically describe their phase transition during the optimization process of Stochastic Gradient Descent (SGD) (see, for example, [9, 10]). Some others have focused instead on the theory of stochastic processes to study the dynamics of the parameters under the SGD mechanism [11–14]. Along the same vein, a more geometrical modeling via thermodynamics concepts appears in [15–17]. Inspired by this literature, in [1, 18, 19] we employed concepts from thermodynamics and molecular dynamics to establish a more stringent analogy between thermodynamic systems and neural networks. In particular, we introduced a new concrete definition of temperature for the parameters of a neural network and we studied it across the layers of different CNN and GNN architectures.

Besides the physically inspired theoretical efforts, a more empirical way to interpret neural networks is through the practice of pruning [20, 21]. Pruning has the aim of removing unimportant weights from a deep learning model to reduce the model's memory footprint and accelerate inference performance. It is well known that neural networks usually work with an excess of parameters with respect to the ones actually needed for a good generalization [22]. This is because, to this day, there are no established theoretical guidelines in assigning the size of a neural network layer, e.g., the kernel size and the number of filters of a CNN or the embedding dimension in a GNN, and those are finally chosen by means of a blind and expensive grid search. Hence, the design of a pruning

^a e-mail: michela.lapenna4@unibo.it (corresponding author)

^b e-mail: francesco.faglioni@unimore.it

^c e-mail: rita.fioresi@unibo.it

^d e-mail: giovanni.bruno@bam.de

technique inspired by mathematical and physical reasoning is crucial to take a step toward a more accurate understanding of neural networks. At the same time, cleverly deciding the size of the model would make deep learning training more sustainable [23].

Common pruning techniques are magnitude-based [24–26] and gradient-based [27–29]. The first kind consists in pruning weights with small magnitude: in [26], for example, the filters of a CNN are pruned depending on the sum of their absolute kernel weights, i.e., their l_1 -norm. This pruning technique is justified by the fact that the smaller the sum of the absolute value of the weights, the weaker their activations compared to other filters in that layer. In contrast, gradient-based pruning assesses weight importance by measuring the sensitivity of the loss to changes in each weight. A classical example is the Optimal Brain Surgeon (OBS) algorithm [28], which states that low-magnitude weights can still be crucial for minimizing error and evaluates the full Hessian of the loss at equilibrium to identify weights whose removal minimally increases training loss. To reduce OBS computational overhead, [29] approximates the change in the loss function caused by pruning network parameters using a first-order Taylor expansion. Pruning is then performed greedily, with gradients computed from a single training step at each iteration.

In this work, we propose a novel gradient-based pruning method inspired by a thermodynamic analogy. This method builds upon our previous work [1], where we exploit the thermodynamic framework introduced in [18] to design a temperature-based pruning technique for CNN filters. Rather than relying on instantaneous gradient snapshots as in [29], temperature is proportional to the average gradient magnitude over equilibrium epochs, when loss and accuracy have reached convergence. Independently on the optimizer employed, weight structures with persistently low gradient prove to be the ones to prune.

Specifically, in the present paper, we employ the temperature-based pruning technique to prune input features in GNNs. The notion of temperature allows us to prune rows of weights in the embedding matrix of the first layer in the GNN architecture, each row directly corresponding to a node feature. We can then understand the key information in the input data, used by the GNN to reach a successful completion of the task at hand. In fact, in state-of-the-art benchmark datasets for node and graph property prediction [30–34], each node comes equipped with a vector of numerous features, usually obtained by extracting information from the data itself, e.g., a dictionary of words for a citation network or biological properties for molecule datasets. It is paramount to understand which information is actually necessary and which can be discarded. This constitutes a key step in understanding how GNNs learn from data and can help in guiding the feature engineering process itself, toward a better explainability of GNNs and efficiency improvement (see [35, 36] for related works).

This paper is organized as follows. In Sec. 2, we give some physical intuition about SGD and we recall our definition of temperature for a neural network [1, 18, 19]. A more extensive explanation of the theoretical basis for our modeling is given in Appendix A. In Sec. 3, we describe the GNN architectures considered together with the datasets employed, and we give details about the training. Then, in Sec. 4, we present the results of the pruning experiments on the input features to the GNN. Finally, in Sec. 5, we draw conclusions. Appendix B compares our pruning approach with a statistical correlation analysis of input features. Appendix C contrasts pruning input features with pruning embedding dimensions and deeper layers. Appendix D compares pruning in GNNs and CNNs. Appendix E provides additional plots on GNN input feature pruning.

2 Temperatures in Neural Networks

We briefly summarize the thermodynamic modeling appearing in [1, 18, 19]. When performing optimization in Deep Learning algorithms, a common technique to minimize the loss function is Stochastic Gradient Descent [37, 38] and its variations (e.g., SGD with momentum [39] and Adam [40]).

Let $\Sigma = \{z_i \mid 1 \leq i \leq N\} \subset \mathbb{R}^D$ denote a dataset of size N , i.e., $|\Sigma| = N$, $L = \frac{1}{N} \sum_{i=1}^N L_i$ the loss function, with L_i the loss of the i -th datum z_i , and $\mathcal{B}$ the batch size. The SGD update for the weights $w = (w_k) \in \mathbb{R}^d$ of the model at each step is:

$$\mathbf{w} \rightarrow \mathbf{w} - \eta \nabla_{\mathcal{B}} L(\mathbf{w}) \quad \text{with} \quad \nabla_{\mathcal{B}} L := \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla L_i \quad (1)$$

where η denotes the learning rate. In [11], the discrete SGD update is rewritten in the continuous time limit via a stochastic differential equation (Ito formalism [41]):

$$d\mathbf{w}(t) = -\eta \nabla L(\mathbf{w}) dt + \sqrt{2\zeta^{-1} D(\mathbf{w})} dW(t) \quad (2)$$

where $W(t)$ is the *Wiener process* modeling the stochasticity of the descent, while $D(\mathbf{w})$ is the *diffusion matrix*, controlling the anisotropy of the diffusivity in the process. The quantity ζ is called the *inverse temperature*, and it is defined as $\zeta^{-1} = \frac{\eta}{2|\mathcal{B}|}$. This parameter captures the amount of noise due to SGD: indeed, small batch sizes or high learning rates cause the trajectories of the weights to be “noisier”. The convergence of the stochastic gradient algorithm to this continuous time stochastic differential equation (SDE) is proven in [42] (see [43] for a Markov chain Monte Carlo (MCMC) sampling treatment). It is important to note that the SDE formulation in equation (2) is strictly valid only in the case of SGD.

In [18], the stochastic behavior modeled by SGD is then accounted for introducing a new concrete measure of temperature for neural networks. The weights are treated as non-quantum and non-relativistic particles, and their scalar value is considered as their

position. The *instantaneous temperature* $\mathcal{T}(t)$ of the system is defined as its kinetic energy $\mathcal{K}(t)$ divided by a constant $k_B > 0$ to obtain the desired units:

$$\mathcal{T}(t) = \frac{\mathcal{K}(t)}{k_B} = \frac{1}{k_B d} \sum_{k=1}^d \frac{1}{2} m_k v_k(t)^2 \quad (3)$$

where d is the number of degrees of freedom (in our case the dimension of the weight space) and $v_k(t)$ is the instantaneous velocity of one parameter, computed as the difference between the value of the k^{th} parameter at one step of training and its value at the previous step (the shift in time Δt is unitary since we are computing the instantaneous velocity between consecutive steps or epochs):

$$v_k(t) = \frac{w_k(t) - w_k(t-1)}{\Delta t} \quad (4)$$

In the formula for the kinetic energy, m_k is the mass of parameter w_k and it is set to 1. This is because we have not modeled yet the role of the mass in describing the dynamics of the parameters during the training. The *thermodynamic temperature* is then the time average of $\mathcal{T}(t)$:

$$T = \frac{1}{\tau} \int_0^\tau \mathcal{T}(t) dt = \frac{1}{\tau k_B} \int_0^\tau \mathcal{K}(t) dt = \frac{K}{k_B} \quad (5)$$

where K is the average kinetic energy, and τ is an interval of time long enough to account for small variation in temperature. Specifically, we train the neural network until the loss and accuracy converge, and we measure the thermodynamic temperature by averaging over epochs of training after convergence has been reached (equilibrium epochs). Instead of focusing on the time steps inside the last epoch of training, we think that it is more representative to work between equilibrium epochs. Indeed, due to the SGD mechanism, the evolution of the velocities between training steps is influenced by the fact that we are looking at a different batch of data at each step.

In the SGD regime, with or without momentum, our definition of thermodynamic temperature is directly proportional to the squared gradient of the loss function (see Appendix A for a complete derivation). For this reason, we generalize our notion of temperature and we call *SGD-temperature* the mean squared gradient of the loss:

$$T_{\text{SGD}} = \frac{1}{N\tau} \sum_{k=1}^N \sum_{t=1}^\tau \left(\frac{\partial L}{\partial w_k}(t) \right)^2 \quad (6)$$

As we shall see, this notion allows for a uniform criterion for pruning across optimizers. We stress that, when pruning, we consider the temperature for different weight structures relative to each other and to the entire architecture. For example, when pruning input features in a GNN, we compare the temperature for the different rows of the embedding matrix in the first layer, thus setting N in eq. (5) and eq. (6) equal to the embedding dimension.

In [1, 18, 19], we measure the temperature in eq. (5) for the layers of some CNN and GNN architectures. In particular, we measure the equilibrium temperature of a layer by averaging on the mean squared velocities of its weights over the equilibrium epochs. In [18] and [19], we study the dependence of temperature with respect to the hyperparameters of η learning rate and B batch size. Different layers exhibit slightly different behaviors in the dependence of η and $\frac{1}{|B|}$ from the temperature and our measure of temperature is not linearly proportional to $\frac{\eta}{|B|}$ as it is stated for ζ^{-1} in equation (2). Furthermore, we observe that the temperature decreases as we go deep inside the architecture, i.e., the first layers have a higher temperature with respect to the last ones [18].

Additionally, if we look at the mean squared velocities of the weights over the equilibrium epochs, without averaging on the number of weights, we observe that the weights inside the same layer show a completely different thermal agitation [1, 19]. We hence consider using temperature at equilibrium as a criterion to distinguish which parameters have been correctly optimized and which have not. However, we stress again that this effect is diminished with network depth and the weights belonging to the last layers, i.e., further from the input, share a more homogeneous excitation pattern. As our experiments show, while different optimizers show different behaviors with respect to pruning when measuring temperature with eq. (5), SGD-temperature in eq. (6) appears a consistent criterion for effective pruning.

In [1], using Adam optimizer, we see that we can prune CNN filters with higher temperature (i.e., mean squared velocity) at equilibrium without affecting the overall performance, while “cold” filters learn useful patterns. For GNNs the story is quite different and indeed reflects the meaning of node features in a GNN: each row of the embedding matrix in the first GNN layer (i.e., closer to the input) correspond to a node feature. In this work on GNNs, we use both SGD and Adam optimizers. While temperature in eq. (5) leads to pruning “hot” structures of weights when using Adam and “cold” ones when using SGD, the SGD-temperature criterion is consistent across both optimizers, and this consistency applies to pruning “cold” structures in both GNNs and CNNs. Additional plots illustrating these results for CNN filter pruning are presented in Appendix D.

We expect our pruning technique to be general and applicable to other types of neural networks and different weight structures. By now, we have focused on CNNs filters and GNNs input features to highlight the explainability power of our technique, besides its practical importance, coming from the reduction of the complexity of the neural network after pruning.

3 Methods

In this study, we analyze two GNN architectures, one to perform link prediction on Cora [30] and Citeseer [31] datasets and the other one to solve a node classification task on Pubmed [33] and PPI [34] datasets (see Fig. 1). For the link prediction task, we employ Graph Attention layers (GAT) [44], while for node classification a GraphSAGE [45] model. We design different GNN architectures for different tasks to prove the generality of our temperature-based pruning technique on the input features in GNNs. For GNN state-of-the-art performances see [46] for link prediction and [47] for node classification.

Cora, Citeseer and Pubmed datasets are single-graph citation networks where each node represents a scientific publication. In Cora and Citeseer, the input feature for the node is a 0/1-valued word vector indicating the absence/presence of the corresponding word in a dictionary extracted from the dataset: the dictionary consists of 1433 unique words in Cora and 3703 in Citeseer. For what concerns Pubmed, each node is described by a TF/IDF weighted word vector from a dictionary of 500 unique words and is classified into one of three classes. TF/IDF, which stands for Term Frequency-Inverse Document Frequency, is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents. On the other side, PPI is a Protein–Protein Interaction dataset consisting of 20 graphs. Each node represents a protein, and the edges in the graph represent the interactions between proteins. The input vector for each node consists of 50 features obtained by dimensionality reduction from a longer vector including biological and biochemical properties of the proteins. Each node can belong to multiple classes, representing different biological functions, and the total number of classes is 102. We test our temperature-based pruning technique on these datasets to distinguish between more and less useful features attached to the nodes, with the potential of reducing the input information needed and explain how this information is used by GNNs in the learning process.

We experiment with both SGD and Adam optimizers, setting the learning rate to 0.1. For SGD, we explore configurations with momentum ($\beta = 0.9$) and with or without L_2 weight decay ($\lambda = 5 \times 10^{-4}$). For Adam, we employ the standard first and second moment estimates ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) and consider setups with or without L_2 weight decay ($\lambda = 5 \times 10^{-4}$). Contrary to what is done in [1, 18, 19], we apply batch normalization in the link prediction architecture (see Fig. 1).

For the link prediction task, since we are solving a binary classification problem on the edges, we use a binary cross-entropy loss. Indeed, we want to predict whether the edge exists (label 1) or does not exist (label 0). The edge embedding is computed by applying the scalar product to the final layer embeddings of the connected nodes, and the likelihood of edge existence is scored via softmax. Also, in order to train and test our binary classifier, the dataset is enriched with non-existent edges with label 0. For node classification, instead, we employ the negative log likelihood loss on the number of classes. For all datasets, the split ratio is of 85% for the training set, 5% for the validation set and 10% for the test set. We stress that in the case of single-graph datasets (Cora, Citeseer and Pubmed), the split is done in a transductive way on the edges (link prediction) or on the nodes (node classification), while for PPI dataset the split is inductive on the number of graphs. The model is trained 10 times with a fixed random seed and for each training we save the weights for the model performing the best on the validation set.

As stated in Sec. 2, we measure the thermodynamic temperature of a neural network by averaging on the equilibrium epochs. We train the models in Fig. 1 until loss and accuracy converge; then, we compute the mean squared velocities in eq. (5), or the mean squared gradients in eq. (6), of the weights over 200 equilibrium epochs. To implement the temperature-based pruning for the input features, we measure the temperature of each input feature by averaging on the output dimension of the embedding matrix of the first GAT and GraphSAGE layers. We test the effects of pruning the input features with two different experiments. First, we just zero-out the entries in the input vectors, by maintaining their original dimension, and we evaluate the model again on the pruned dataset. Then, we suppress the features by reducing the dimensionality of the input vectors, and we retrain the model from scratch on the new graph. Furthermore, we compare our pruning criterion with a simple magnitude-based approach [26], to investigate whether magnitude is also a consistent indicator for pruning in GNNs. In Appendix B, we also compare our temperature-based criterion with a straightforward statistical correlation analysis of input features.

Fig. 1 GNN architectures used for link prediction (Cora and Citeseer) and node classification (Pubmed and PPI)

Link Prediction:

- GATConv(input $\rightarrow$ 64, heads=2)
- BatchNorm1d(64 * 2)
- LeakyReLU
- GATConv(64 * 2 $\rightarrow$ 128, heads=2)
- BatchNorm1d(128 * 2)

Node Classification:

- SAGEConv(input $\rightarrow$ 64)
- Dropout $p = 0.05$
- LeakyReLU
- SAGEConv(64 $\rightarrow$ 128)
- Dropout $p = 0.05$
- LeakyReLU
- SAGEConv(128 $\rightarrow$ classes)

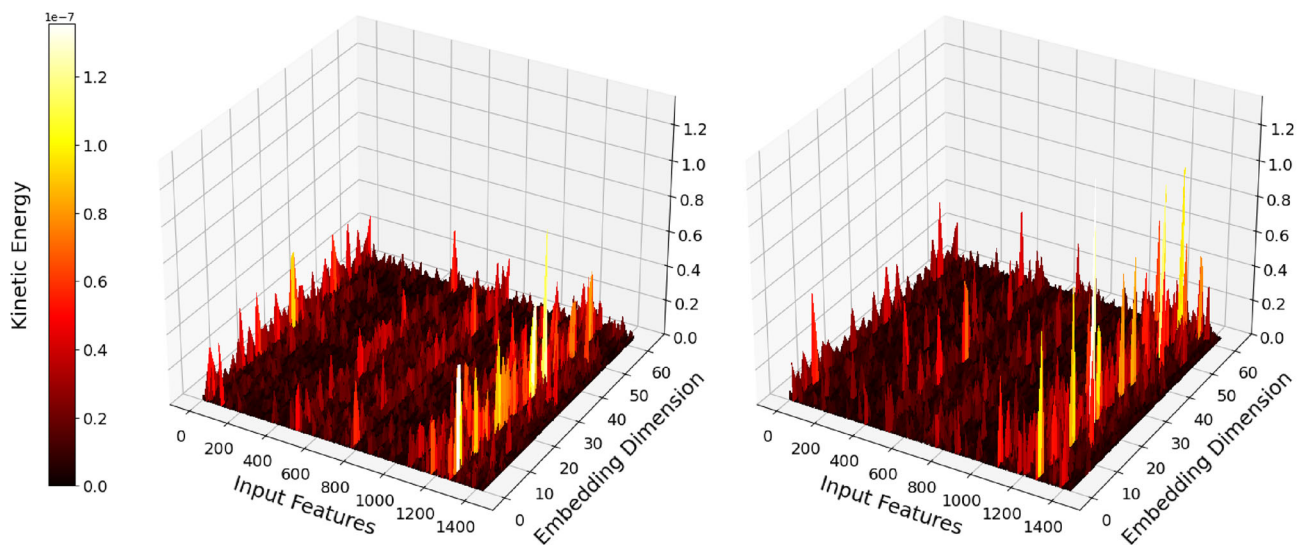


Fig. 2 Thermal agitation at equilibrium for the two heads of the first GAT layer in the case of Cora dataset. These heatmaps correspond to a single training with SGD with momentum and without weight decay

4 Results

We describe the results obtained when pruning the input features for Cora [30], Citeseer [31], Pubmed [33] and PPI [34] datasets. From the sketch of the link prediction architecture in Fig 1, we see how each head of the first GAT layer embeds the vector of features for each node from its input dimension (1433 in Cora and 3703 in Citeseer) to a dimension of 64. Similarly, in the node classification case, the first GraphSAGE layer embeds Pubmed and PPI, respectively, 500-dimensional and 50-dimensional input vectors to a dimension of 64.

If we measure the mean kinetic energy of the weights belonging to the embedding matrix of the first GAT and GraphSAGE layers, we notice that some features in the input vector induce higher thermal agitation than others (see Fig. 2 for a visualization in the case of Cora). Hence, we extract the temperature for each input feature by averaging on the output dimension of the first GNN layer and we use this as the criterion to prune useless input features. In the case of GAT layer, we average on both heads.

The first pruning experiment consists in zeroing-out a varying number of entries (features) in the input vector of each node, going from the “hottest” to the “coldest” and vice versa. After pruning, we evaluate again the model (on the pruned dataset) and we look at how the performance has changed with respect to the original setting. In Fig. 3, we present the pruning curves obtained on Cora using the following different criteria:

1. the SGD-temperature (eq. 6), proportional to the mean squared gradient of the loss;
2. the temperature defined as proportional to the mean kinetic energy (eq. 5);
3. the magnitude, i.e., the mean absolute value of the weights.

We stress that when measuring the magnitude of each feature, we first average on the equilibrium epochs and then on the output dimension of the GNN embedding matrix, as for temperature and SGD-temperature.

Our analysis shows that, based on the temperature criterion (eq. 5), SGD generally selects “cold” features for pruning, whereas Adam tends to select the “hot” ones. A more detailed analysis of this phenomenon is provided in Appendix A. However, these differences reconcile once we take into exam the SGD-temperature criterion (eq. 6). SGD-temperature yields consistent results across different optimizers: the features selected for pruning are those associated with lower gradient magnitudes. On the other hand, magnitude proves to be an inconsistent indicator of feature importance across different optimizers and does not always suggest pruning features with low magnitude, as demonstrated for CNN filters in [26].

Very similar results to those shown in Fig. 3 are observed for Citeseer dataset, and we provide the corresponding plots in Appendix D. In our experiments on both Cora and Citeseer, where input features are binary and highly sparse, we find that pruning the least important features does not significantly impact performance until a large proportion of features has been removed. We attribute this to the nature of the input features in these two datasets: feature vectors are represented as bag-of-words, with each dimension being binary (1 indicating the presence of a word and 0 its absence). Due to the high sparsity (about 90% of zeros), many features are effectively inactive (zero) and, as shown in Appendix B, those are usually the ones deemed as useless. Consequently, pruning these features typically amounts to zeroing out elements that were already inactive, causing minimal performance impact. In contrast, pruning highly useful (active or nonzero) features leads to rapid degradation of performance.

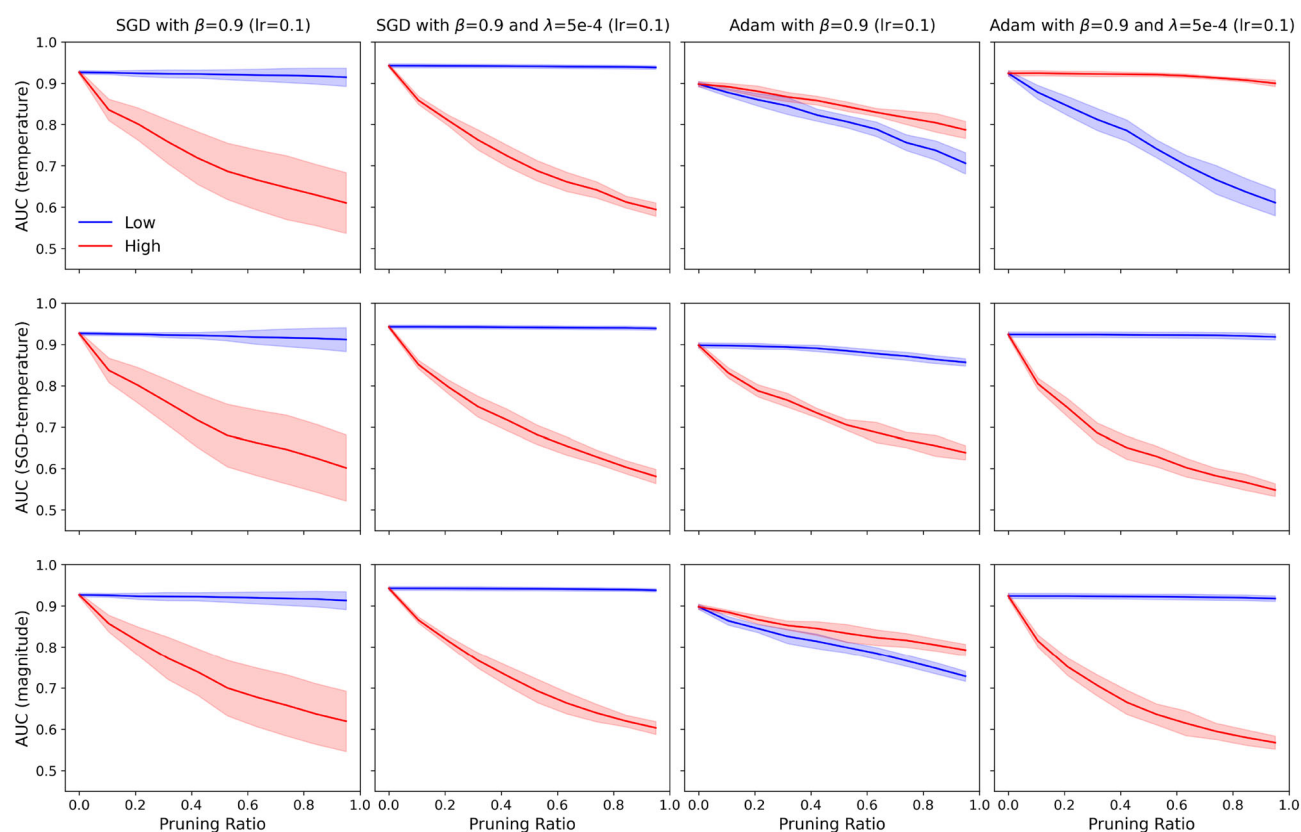


Fig. 3 Performance after zeroing-out features and re-evaluating the model on the pruned test set based on temperature, SGD-temperature and magnitude (Cora dataset). Features are pruned by increasing ratio from the “hottest” to the “coldest” (red curve) and vice versa (blue curve). Shaded areas show standard deviation over 10 trained models. AUC indicates area under the curve

For the node classification scenario, as shown in Fig. 4 for Pubmed and PPI datasets, we observe once again that performance remains closer to the original when pruning the features with low rather than high SGD-temperature. However, when pruning “colder” features, the performance degrades more rapidly in Pubmed and PPI compared to Cora and Citeseer.

Although the feature vectors in Pubmed are also highly sparse (about 90% of zeros), the input dimensionality in Pubmed and PPI datasets is considerably lower than that in Cora and Citeseer. Notably, the gap between the “hot” and “cold” pruning curves is especially narrow in PPI dataset (see Fig. 4), where the input vectors have a dimensionality of just 50, and all features seem to be important for the node classification task. Furthermore, in PPI, the features are real-valued rather than sparse binary, which may further contribute to the observed behavior.

Additionally, we broaden our analysis by applying the temperature-based pruning to reduce the dimensionality of the input feature vectors, instead of simply zeroing out features, and then retraining the model from scratch on the pruned dataset. This process is repeated for progressively larger sets of features, removing them in order from “hottest” to “coldest” and vice versa. The model is retrained following the procedure described in Section 3, and we compare its new performance to the original. It is important to highlight that reducing the dimensionality of the input vectors also reduces the size of the embedding matrices in the first GAT and GraphSAGE layers, thereby decreasing the total number of trainable parameters in the model.

In Fig. 5, we present the performance curves on Cora and Pubmed datasets, when pruning based on SGD-temperature. In line with our initial pruning experiment, we find that pruning “colder” features has little effect on performance, whereas pruning “hot” features causes a more substantial degradation. Even in this scenario, we observe that a substantial portion of “cold” features can be pruned without significantly affecting performance in the highly sparse Cora and Pubmed datasets. Moreover, compared to the case where the model is not retrained, pruning “hot” features now leads to a smaller performance drop. We attribute this improvement to the model’s ability to adapt to the pruned dataset through retraining.

This further demonstrates that the model can maintain strong performance even with a greatly reduced input feature dimension, although performance inevitably declines when the feature dimensionality becomes very low (as also observed in [49, 50]). These observations are consistent with prior work on pruning GNN input features [35], which showed that in node classification tasks with GNNs on datasets like Cora, Citeseer, and Pubmed, accuracy decreases by only about 15% even after reducing the feature dimensionality by over 80%.

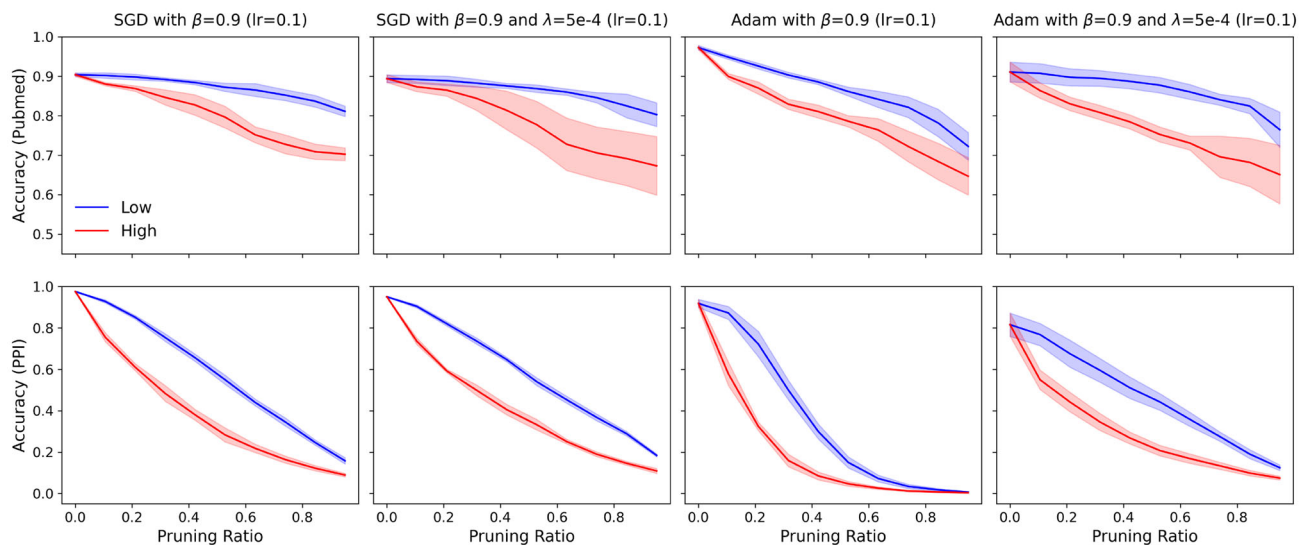


Fig. 4 Performance after zeroing-out features and re-evaluating the model on the pruned test set based on SGD-temperature (Pubmed and PPI datasets)

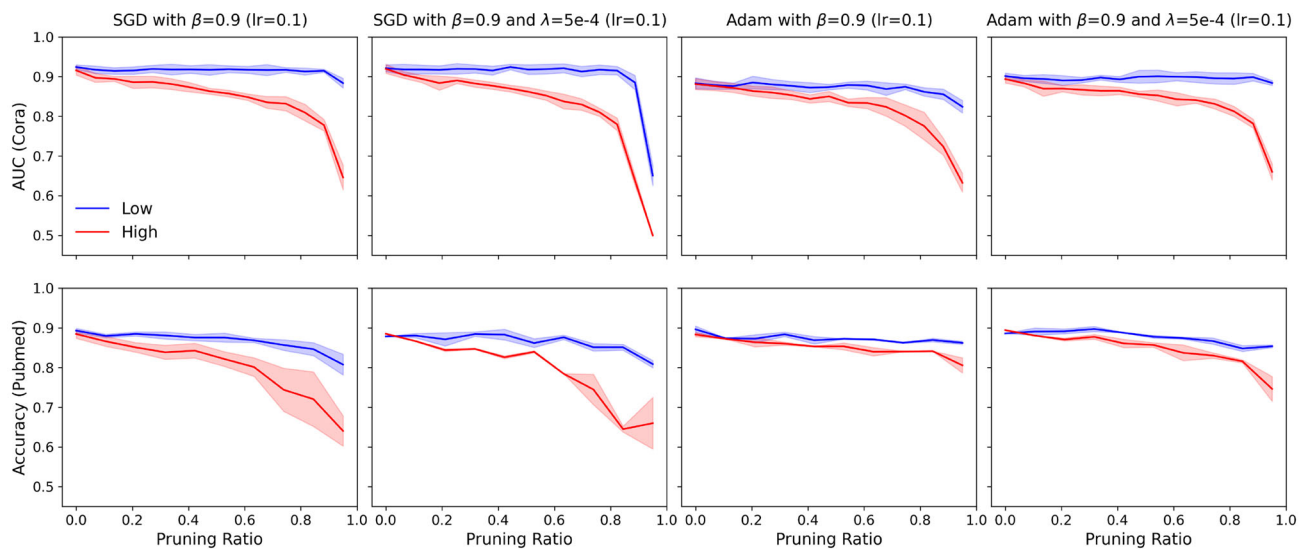


Fig. 5 Performance after re-training and re-evaluating the model on the pruned dataset based on SGD-temperature (Cora and Pubmed datasets)

5 Conclusions

We employ a temperature-based pruning technique to highlight which of the features contained in the node vectors in input to a GNN are important and which can be discarded. Our generalized notion of temperature, called SGD-temperature and defined as the mean squared gradient of the loss function, is consistently pruning the “cold” structures of weights, independently of which optimizer is employed. Our experiments on several benchmark datasets (Cora [30], Citeseer [31], Pubmed [33], PPI [34]) and different GNN architectures (GAT [44], GraphSAGE [45]) show that “hot” features, i.e., the ones inducing higher gradients, are essential for the link prediction and node classification tasks under analysis. We compare the results of these pruning experiments to our previous experiments on the filters of a CNN [1] (see Appendix D). Moreover, we extend our analysis by applying the SGD-temperature criterion to prune columns of the GNN embedding matrix, effectively reducing the embedding dimension of the network (see Appendix C). Overall, our findings highlight that considering only the magnitude of weights can be misleading when identifying essential structures in neural networks. By incorporating a dynamics measure such as the SGD-temperature, we gain a deeper understanding of the network’s learning process and its convergence behavior, ultimately improving the interpretability and effectiveness of pruning strategies.

Acknowledgements We are indebted to our Referees for many valuable comments that helped us to improve our paper. We thank Ferdinando Zanchetta and Augustin Somacal for valuable suggestions. The research ML and RF was supported by COST Action CaLISTA CA21109, GNSAGA-Indam, PNRR

MNESYS, PNRR National Center for HPC, Big Data and Quantum Computing CUP J33C22001170001, PNRR SIMQuSEC CUP J13C22000680006, by HORIZON-MSCA-2022-SE-01-101086123 CaLiGOLA and by HORIZON-MSCA-DN CaLiForNIA, Project ID: 101119552. Funding for ML from BAM Project QI Digital is acknowledged. FF acknowledges UNIMORE FAR-DIP DSCG 2023.

Author Contributions ML contributed to conceptualization, data curation, formal analysis, investigation, methodology, and writing—original draft. FF, RF, GB contributed to supervision and writing—review and editing.

Funding Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement.

Data Availability Cora [30], Citeseer [31], Pubmed [33] and PPI [33] datasets are imported from PyTorch Geometric and we handle the transductive/inductive split of the graph/graphs with DeepSNAP library.

Declarations

Ethical Approval Not applicable.

Conflict of interest The authors declare that no Conflict of interest may have influenced the work in this paper.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

A Thermodynamics modeling of Neural Networks weight dynamics

In this appendix, we briefly interpret the behavior of the weights of a neural network (NN) with a thermodynamics formalism, appeared in a series of articles [1, 18, 19]. Although similar interpretations have been published [9–14], a general convincing theoretical understanding for these systems is still lacking. While one must be careful not to rely on the analogies between NNs and thermodynamical systems, this introduces an additional level of analysis with the focus of enhancing interpretability.

The starting point for our thermodynamic modeling is the observation that when a NN is trained with Stochastic Gradient Descent (SGD) [37, 38], or derived methods such as SGD with momentum [39] and Adam optimizer [40], the NN parameters change at each step as the loss function is minimized. Once the NN is trained, the loss stops decreasing and the performance of the NN no longer improves. If training is pushed beyond this point, however, the parameters continue to change at each step, due to the stochastic nature of SGD. This behavior resembles that of individual atomic or molecular positions once thermodynamic equilibrium is reached. To explore this parallelism, we regard the NN as a thermodynamic system, each NN parameter as a coordinate, and the loss as a thermodynamic potential which drives the system toward equilibrium.

A.1 The introduction of an SGD-temperature

In classical dynamics simulations, forces are commonly regarded as gradients of a potential V and coordinates evolve according to Newton's law:

$$F_k = -\frac{\partial V}{\partial x_k} \quad a_k = \frac{d^2 x_k}{dt^2} = \frac{F_k}{m_k} \quad (7)$$

When discretized for numerical integration, coordinate evolution at each time-step t takes the form:

$$\begin{aligned} x_k(t) &= x_k(t-1) + v_k(t-1)\Delta t + \frac{1}{2}a_k(t-1)\Delta t^2 = \\ &= x_k(t-1) + v_k(t-1)\Delta t - \frac{1}{2m_k} \left(\frac{\partial V}{\partial x_k} \right)_{t-1} \Delta t^2 \end{aligned} \quad (8)$$

where m_k is the mass associated with each degree of freedom, and $v_k(t)$ is the instantaneous velocity. This expression describes constant energy dynamics. The instantaneous temperature $\mathcal{T}(t)$ is defined as the mean kinetic energy:

$$\mathcal{T}(t) = \frac{\mathcal{K}(t)}{k_B} = \frac{1}{k_B d} \sum_{k=1}^d \frac{1}{2} m_k v_k(t)^2 \quad (9)$$

where $\mathcal{K}(t)$ is the (instantaneous) kinetic energy, d is the number of degrees of freedom, and k_B is Boltzmann's constant. The thermodynamic quantities are obtained as time-averages of the corresponding instantaneous forms. For average kinetic energy K and temperature T we have:

$$K = \frac{1}{\tau} \int_0^\tau \mathcal{K}(t) dt = \frac{1}{\tau} \sum_{t=0}^\tau \mathcal{K}(t) \quad (10)$$

$$T = \frac{K}{k_B} \quad (11)$$

where the averages are computed over a long enough sampling time τ to ensure fluctuations in the resulting values are negligible.

To perform constant temperature dynamics, contact with a heat reservoir is mimicked by periodically rescaling the velocity of each particle in order to adjust the average kinetic energy to the desired temperature, leading to the expression:

$$x_k(t) = x_k(t-1) + \beta v_k(t-1) \Delta t - \frac{1}{2m_k} \left(\frac{\partial V}{\partial x_k} \right)_{t-1} \Delta t^2, \quad (12)$$

where $\beta > 1$ ($\beta < 1$) is used to warm (cool), i.e., to transfer heat into (out of) the system.

In the SGD algorithm for NNs, both with and without momentum, parameters evolve according to the gradient of the loss function L :

$$f_k(t) = \beta f_k(t-1) + \eta \left(\frac{\partial L}{\partial w_k} \right)_{t-1} \quad w_k(t) = w_k(t-1) - f_k(t) \quad (13)$$

where $\beta = 0$ describes SGD without momentum. This corresponds to a discretized parameter velocity:

$$v_k(t) = w_k(t) - w_k(t-1) = -f_k(t) = \beta v_k(t-1) - \eta \left(\frac{\partial L}{\partial w_k} \right)_{t-1}. \quad (14)$$

or

$$w_k(t) = w_k(t-1) + \beta v_k(t-1) - \eta \left(\frac{\partial L}{\partial w_k} \right)_{t-1} \quad (15)$$

which is similar to eq. (12) above, if we let $\Delta t = 1$, $\eta = \frac{1}{2m_k}$, and identify L with a conservative potential driving the dynamics. Since in NN optimizations $\beta \in [0, 1]$, the evolution of the NN parameters driven by loss minimization is analogous to the evolution of particles driven by potential minimization when the particle velocities are quenched at each time-step, i.e., when heat is extracted from the system. In molecular dynamics, this leads to constant temperature dynamics, which does not minimize the average energy but the Helmholtz or Gibbs potential, depending on volume or pressure constraints (see [15] for a Lie group thermodynamics treatment).

Since there is no obvious way to reduce the number of degrees of freedom d , we set d equal to the number of parameters. Hence, we associate each parameter to a coordinate. Also, for the time being, we assume that all parameters have the same mass, set to $m_k = 1$. This assumption is not necessary, but assigning masses would require further assumptions, such as thermal equilibrium within the system, leading to a less straightforward description. This leads to the slightly simplified form for the NN temperature, equivalent to the one in eq. (9) for a thermodynamical system:

$$\mathcal{T}(t) = \frac{\mathcal{K}(t)}{k_B} = \frac{1}{k_B N} \sum_{k=1}^N \frac{1}{2} v_k(t)^2 \quad (16)$$

where N is the number of parameters and k_B is a constant to provide the correct physical dimensions, equivalent to Boltzmann's constant in thermodynamics. Velocities are computed as variations between steps of the corresponding parameters w :

$$v_k(t) = w_k(t) - w_k(t-1) \quad (17)$$

In the case of SGD, eq. (17) corresponds to eq. (14), implying that the temperature defined in eq. (16) directly measures the mean squared gradient of the loss.

We experimented with both time-steps between parameter updates and time-steps between epochs. Of course the two average temperatures are proportional to each other, and we choose to study the dynamics of weights between epochs. In our experiments, we measure temperature over equilibrium epochs, when loss and accuracy have reached convergence. Even when the performance no longer improves, usually the gradient of the loss with respect to the parameters is not exactly zero and the weights continue to change at each step. Specifically, we observe that different groups of weights inside the same NN layer show different thermal agitation at equilibrium, and we propose using temperature as a pruning criterion: useless parameters, which consistently receive small gradients, exhibit low temperature, while useful ones remain hot due to larger gradients. As reported in Sec. 2, the temperature of each group of weights is used to identify those that can be pruned. For instance, for input features in a GNN, the temperature for each row of the first layer embedding matrix is considered. In this case, N in eq. (16) is set equal to the embedding dimension.

The same applies when weight regularization is added to the loss:

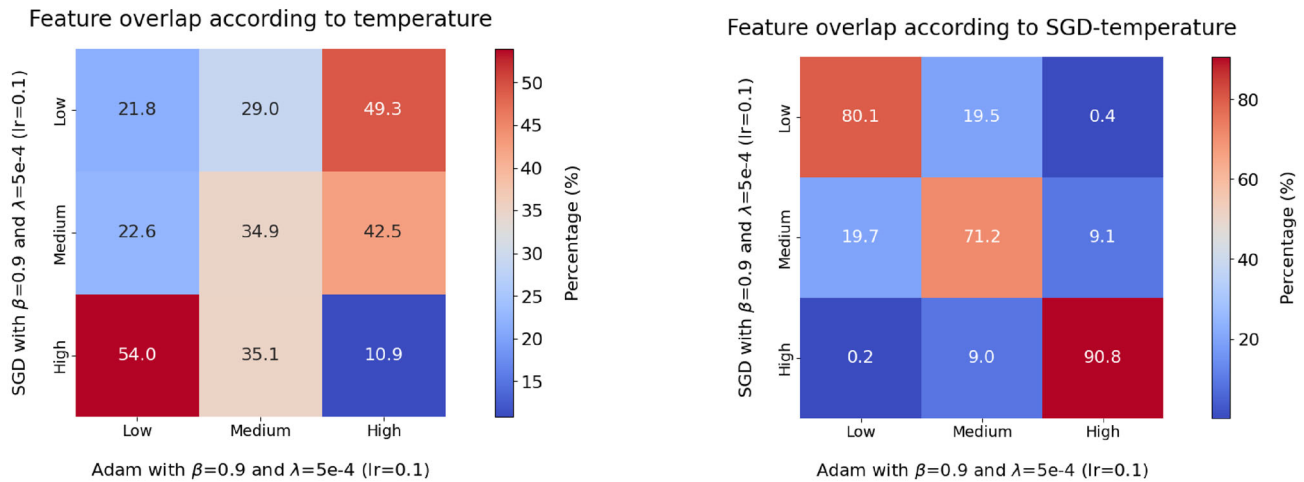


Fig. 6 Heatmaps showing the overlap of Cora input features grouped into three quartiles of temperature in SGD and Adam, each feature corresponding to a row of weights in the GNN embedding matrix. On the left, the temperature calculated using eq. (16), while on the right, the SGD-temperature defined in eq. (21). Results from a single training

$$L_{\text{reg}}(w) = L(w) + \frac{\lambda}{2} \sum_{k=1}^N w_k^2,$$

where λ is the weight decay factor. This term acts like an additive conservative force, pulling the parameters toward the origin.

Adam further alters this picture by incorporating both momentum $\hat{f}_k$ and per-parameter adaptive scaling [40], resulting in updates normalized by the square root of the gradient variance $\hat{s}_k$ (second moment) in the denominator:

$$w_k(t) = w_k(t-1) - \eta \frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t) + \epsilon}}, \quad (18)$$

therefore

$$v_k(t) = w_k(t) - w_k(t-1) = -\eta \frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t) + \epsilon}}, \quad (19)$$

where

$$f_k(t) = \beta_1 f_k(t-1) + (1 - \beta_1) \frac{\partial L}{\partial w_k}, \quad s_k(t) = \beta_2 s_k(t-1) + (1 - \beta_2) \left(\frac{\partial L}{\partial w_k} \right)^2,$$

and

$$\hat{f}_k(t) = \frac{f_k(t)}{1 - \beta_1^t}, \quad \hat{s}_k(t) = \frac{s_k(t)}{1 - \beta_2^t}$$

with standard values $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

In [51], where the sign and magnitude of Adam's update are studied separately, the update magnitude is stated to be governed by the global learning rate η and the factor:

$$\frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)}} = \sqrt{\frac{1}{1 + \frac{\hat{s}_k(t) - \hat{f}_k(t)^2}{\hat{f}_k(t)^2}}}, \quad (20)$$

where $\hat{s}_k(t) - \hat{f}_k(t)^2$ is interpreted as an estimate of the stochastic gradient variance and $\frac{\hat{s}_k(t) - \hat{f}_k(t)^2}{\hat{f}_k(t)^2}$ is called *relative variance*. With respect to SGD, the factor $\frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)}}$ is proved to be responsible for shortening the update in directions of highly relative variance.

A detailed comparison of SGD and Adam is beyond the scope of this article. We observe, however, that Adam forces higher velocities on less informative parameters, even if they receive smaller gradients, disrupting the analogy with statistical thermodynamics. This effect is illustrated in Fig. 6, which shows how groups of weights identified as “cold” under SGD appear “hot” under Adam and vice versa.

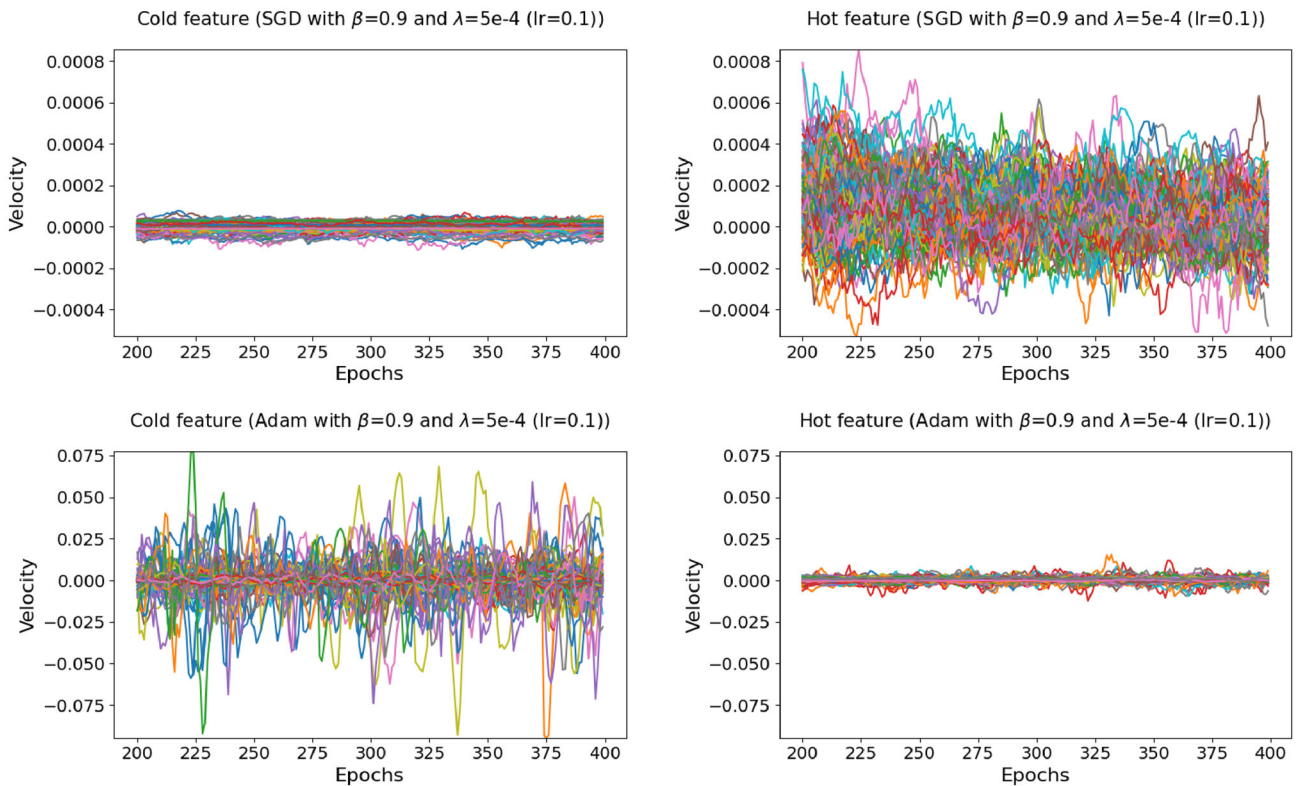


Fig. 7 Velocity of row weights corresponding to the “coldest” (left) and “hottest” (right) Cora input features for SGD (top) and Adam (bottom)

Since in the SGD regime, with or without momentum, our definition of thermodynamic temperature in eq. (16) is directly proportional to the squared gradient of the loss function, we generalize our notion of temperature and we call *SGD-temperature* the mean squared gradient of the loss (same as eq. (6) in Sec. 2):

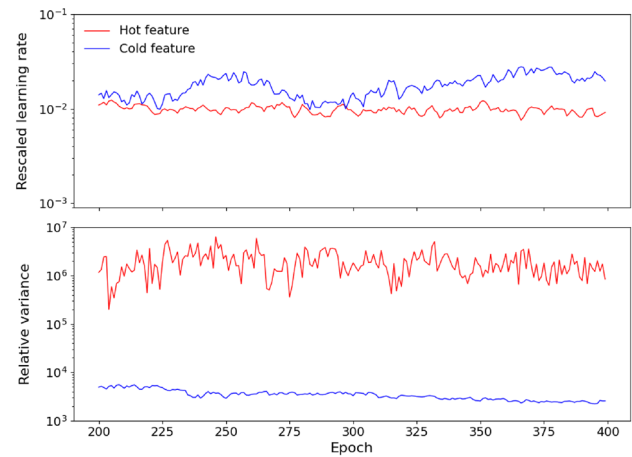
$$T_{\text{SGD}} = \frac{1}{N\tau} \sum_{k=1}^N \sum_{t=1}^{\tau} \left(\frac{\partial L}{\partial w_k}(t) \right)^2 \quad (21)$$

Under the SGD-temperature, weight structures identified as “cold” by SGD correspond to “cold” structures in Adam, and similarly for the “hot” ones (see Fig. 6). Indeed, as shown in Sec. 4, the SGD-temperature serves as a consistent criterion for effective pruning across optimizers.

Furthermore, in Fig. 7, we plot the velocities for the weights in a single row of the GNN embedding matrix over 200 equilibrium epochs, specifically the row corresponding to the feature identified as the “coldest” or “hottest” based on the SGD-temperature at equilibrium. As expected, under SGD the “cold” feature induces lower velocities compared to the “hot” one, whereas under Adam the opposite is observed.

We trace back this behavior to Adam’s scaling factor $\frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)}}$, which, as shown in Eq. (19), is proportional to the velocity of the weight update under Adam. In contrast, for SGD, the velocity is directly proportional to the gradient and thus to the square root of the SGD-temperature. In Fig. 8, we report the absolute rescaled learning rate $\frac{\hat{f}_k(t)}{\sqrt{\hat{s}_k(t)}}$ and the relative variance $\frac{\hat{s}_k(t) - \hat{f}_k(t)^2}{\hat{f}_k(t)^2}$ along the equilibrium epochs for the “coldest” and “hottest” features. At each epoch, these values are obtained by averaging on the entries of the row corresponding to the feature, i.e., the embedding dimension. As expected, the rescaled learning rate is consistently higher for the “coldest” feature, while the relative variance is more than two orders of magnitude larger for the “hottest” one. This indicates that Adam assigns larger rescaled learning rates, i.e., larger velocities, to “colder” parameters. In turn, this suggests that the relative variance in Eq. (20) may actively contribute to amplifying updates for less informative features (with small gradients and low SGD-temperature) and suppressing them for more informative ones. We hypothesize that this mechanism underlies the discrepancy between SGD and Adam when using Eq. (14) to define temperature, and we leave a deeper investigation of this effect to future work.

Fig. 8 Absolute rescaled learning rate and relative variance for the “coldest” (blue) and “hottest” (red) Cora input features when training with Adam with weight decay



A.2 Drift and diffusion in SGD dynamics

To move a step further in the understanding of NN’s behavior, in [1], where we use the Adam optimizer, we also analyze the dynamics of the weights by computing their mean squared displacement (MSD) in time. For each epoch shift, we measure the average displacement of each weight from its initial position. Initially, we expected that, upon convergence, the weights would exhibit Brownian motion around their stable positions due to noisy oscillations. However, we find that even at equilibrium, the weights exhibit a slow drift. According to Brownian motion theory, the MSD is related to time by the following expression:

$$\text{MSD} = 2dDt \quad (22)$$

where d is the dimensionality of the system and D is the diffusion constant. In our case, the system’s dimensionality is 1, as each weight can be seen as a random walk in one dimension. However, this is not a simple random walk since the step size is not necessarily uniform and the probability of moving in either direction is not equal. The dynamics of the weights is influenced by a potential that depends on all other weights, and at each training step, the weight update depends on its position in the parameter space. Nonetheless, the MSD is still related to time, with a different diffusion constant for each weight.

In Brownian motion, MSD is linearly proportional to time, valid asymptotically. For NNs, however, MSD is quadratic in time [1], indicating that the mean displacement is linear in time. This suggests that even at equilibrium, the weights do not exhibit pure Brownian motion but have a residual translational dynamics.

In conclusion, the dynamics of NN weights is characterized by two components: one representing drift or translation and the other representing noisy oscillation. This aligns with the two-term potential introduced by Chaudhari and Soatto in [11], where the discrete SGD update is expressed as a stochastic differential equation (Ito formalism [41]) in the continuous time limit:

$$d\mathbf{w}(t) = -\eta \nabla L(\mathbf{w})dt + \sqrt{2\zeta^{-1}D(\mathbf{w})}dW(t) \quad (23)$$

Here, $W(t)$ accounts for the stochasticity of the descent and models a Wiener process, while $D(\mathbf{w})$ is the diffusion matrix, controlling the anisotropy of the diffusivity. In [11], ζ is called the *inverse temperature*, defined as $\zeta^{-1} = \frac{\eta}{2|B|}$. This parameter captures the noise introduced by SGD, since smaller batch sizes or higher learning rates lead to a “noisier” trajectory in parameter space. Due to SGD’s anisotropy, the authors in [11] suggest that during neural network training, an implicit potential ϕ with different critical points is minimized instead of the original loss function L . This implies that SGD derives the final weight distribution based on the training data distribution, but for a different potential than the one used for back-propagation. The weight distribution ρ is stated to evolve according to the following Fokker–Planck equation:

$$\frac{\partial \rho}{\partial t} = \nabla(\nabla_B L(\mathbf{w})\rho) + \zeta^{-1} \nabla(D(\mathbf{w})\rho) \quad (24)$$

with a unique steady-state distribution satisfying $\frac{\partial \rho^{ss}}{\partial t} = 0$. The term $\nabla(\nabla_B L(\mathbf{w})\rho)$ describes gradient descent dynamics, while $\zeta^{-1} \nabla(D(\mathbf{w})\rho)$ models descent stochasticity. The system of parameters is influenced by the force minimizing the loss function and by the interactions in the parameter space. Indeed, updating weights by examining different dataset portions at each step makes the dynamics stochastic, akin to environmental interactions. The potential $\phi(\mathbf{w})$ is implicitly defined by the steady-state solution ρ^{ss} :

$$\rho^{ss}(\mathbf{w}) = \frac{1}{Z(\zeta)} e^{-\zeta \phi(\mathbf{w})}, \quad \phi(\mathbf{w}) = -\zeta^{-1} \log \rho^{ss}(\mathbf{w}) + \text{constant} \quad (25)$$

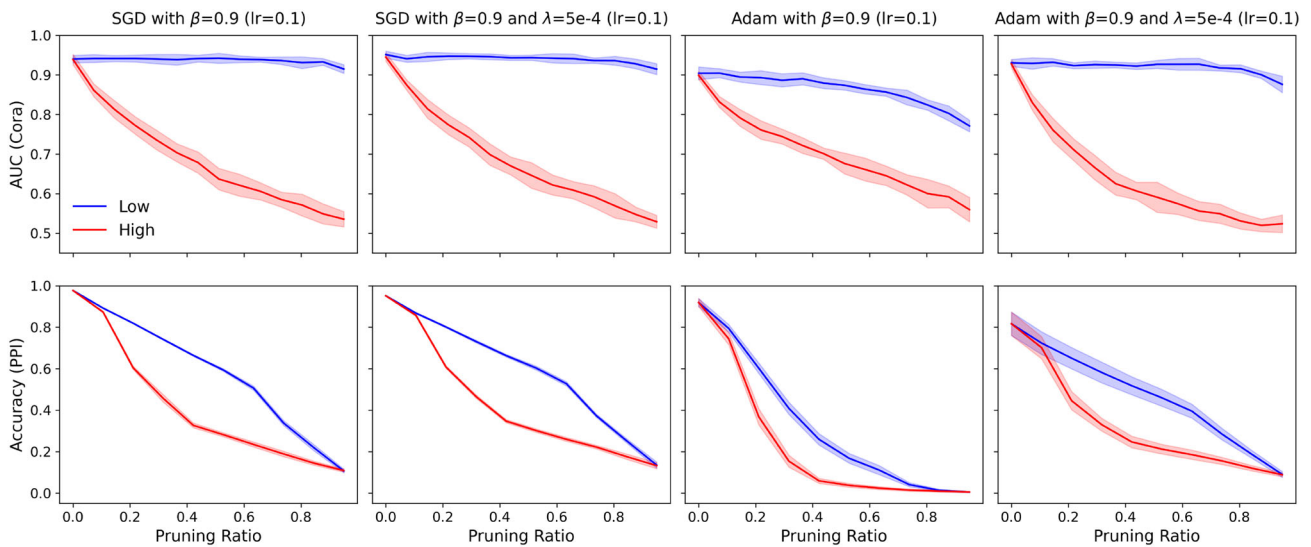


Fig. 9 Performance after zeroing-out features and re-evaluating the model on the pruned test set based on redundancy (Cora and PPI datasets)

where $Z(\zeta)$ is a normalizing function, resembling the partition function from statistical mechanics. Then, SGD implicitly minimizes a combination of “energetic” and “entropic” terms. We associate the translation (or drift) component of the dynamics with minimizing the “energetic” term of the potential, while the oscillation component captures the noisy movement due to the “entropic” term.

Also, to make a parallelism with ζ^{-1} , in [18] and [19] we studied the dependence of our measure of temperature in eq. (16) with respect to the hyperparameters of η learning rate and B batch size. Different layers exhibit slightly different behaviors in the dependence of η and $\frac{1}{|B|}$ from the temperature. Generally, temperature results to be almost proportional to $\frac{1}{|B|}$, while it is linked to η by a quadratic relation. Overall, our measure of temperature is not linearly proportional to $\frac{\eta}{|B|}$ as ζ^{-1} .

B Redundancy through cosine similarity

To assess feature redundancy within the dataset, we construct a redundancy matrix by computing the pairwise cosine similarity between feature vectors $\mathbf{f}_i$ and $\mathbf{f}_j$, where each $\mathbf{f}_i \in \mathbb{R}^{1 \times N}$ represents a feature across all N nodes. The cosine similarity between two features is given by:

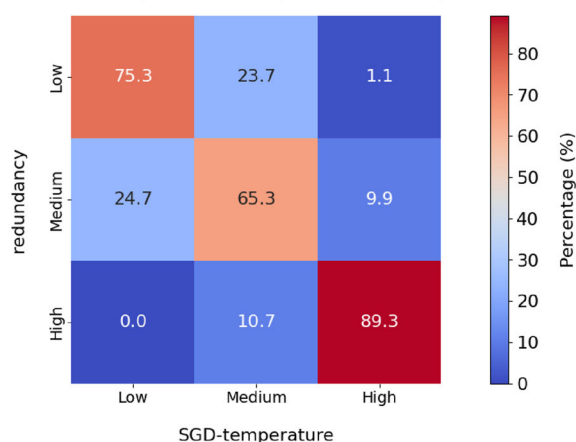
$$\frac{\mathbf{f}_i \cdot \mathbf{f}_j}{\|\mathbf{f}_i\| \|\mathbf{f}_j\|}$$

The resulting redundancy matrix has shape $F \times F$, where F is the total number of features. To quantify how redundant a given feature is, we compute the average cosine similarity between that feature and all the others by averaging on its corresponding row in the redundancy matrix, excluding the diagonal (i.e., self-similarity is ignored by setting diagonal entries to zero). This approach works for both binary and real-valued features, offering a robust measure of redundancy across the dataset.

In Fig. 9, we present the pruning curves derived from redundancy analysis on Cora and PPI datasets. These curves are generated by progressively zeroing out features, from highest to lowest redundancy and vice versa, and subsequently re-evaluating the model on the pruned test set. Interestingly, both figures reveal that pruning low-redundancy features, i.e., those that are more unrelated to others, results in better performance. Notably, these pruning trends closely mirror those observed when using SGD-temperature, as shown in Figs. 3 and 4. It is important to note, however, that redundancy analysis applies strictly to input features, whereas pruning based on T_{SGD} is more general and can be applied across all weight structures, as further discussed in Appendix C (embedding dimensions in GNNs) and Appendix D (CNN filters). Nonetheless, it is reassuring that both approaches seem to select similar sets of features.

As shown in Fig. 10 for the Cora dataset, features classified as “cold” by T_{SGD} generally align with those exhibiting lower redundancy. The same figure further reveals that low redundancy features also tend to be low frequency. Since the feature vectors in Cora are binary, we quantify feature frequency by computing the average value of $\mathbf{f}_i$. This average effectively measures the proportion of nonzero entries in each $\mathbf{f}_i$. Consequently, low frequency features are those that are predominantly zero throughout the dataset. This observation suggests that, in Cora, features which are often zero tend to be the less informative.

Feature overlap redundancy vs. SGD-temperature



Feature overlap redundancy vs. frequency

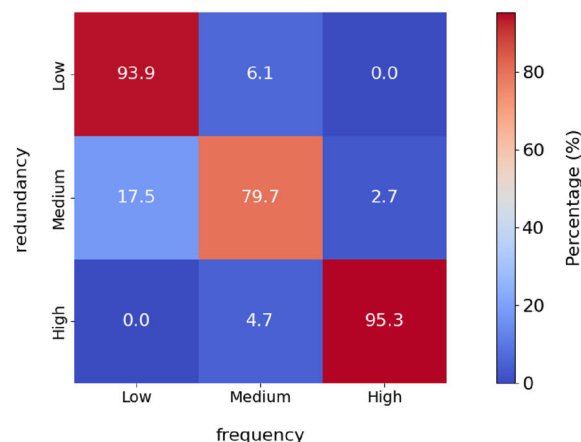


Fig. 10 Heatmaps showing the overlap of Cora input features grouped into quartiles: redundancy vs. SGD-temperature (left) and redundancy vs. frequency (right)

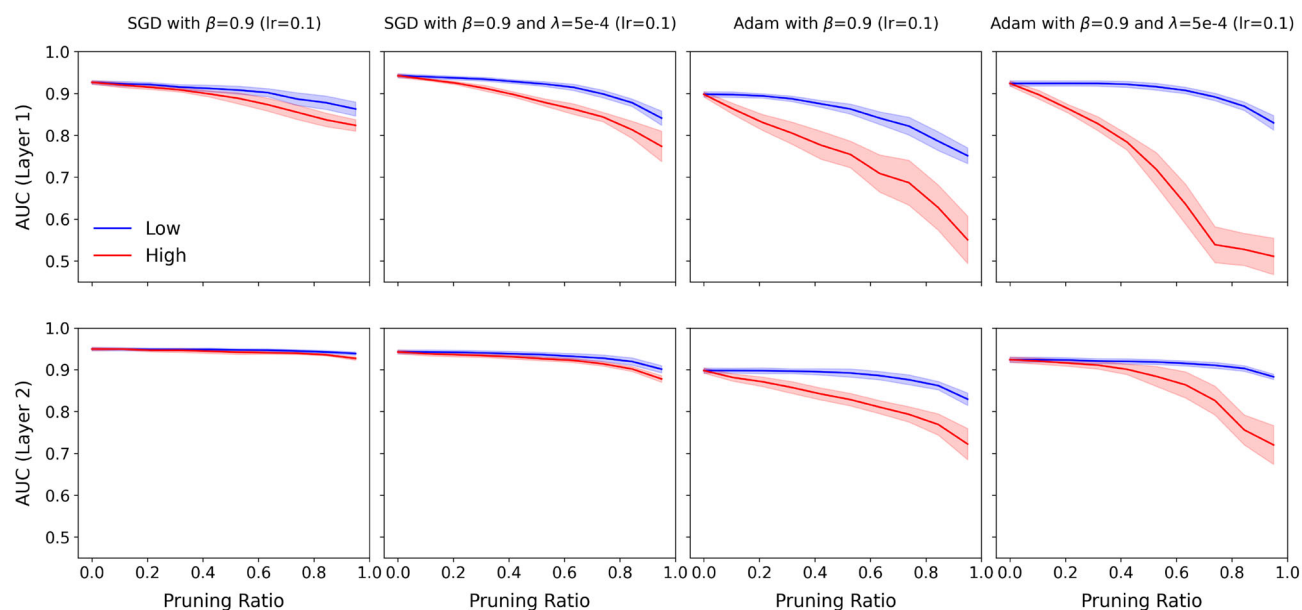


Fig. 11 Performance after zeroing-out embedding dimensions in the first or second GNN layers and re-evaluating the model on the pruned test set based on SGD-temperature (Cora dataset)

C Pruning embedding dimensions

After pruning rows in the GNN embedding matrix, each corresponding to a feature, we further extend the analysis by pruning columns, effectively reducing the embedding dimension of the GNN layer. This is done by computing the SGD-temperature of columns of weights, averaging this time over the input dimension. We show the corresponding pruning curves for Cora dataset in Fig. 11, and for Pubmed and PPI datasets in Fig. 12.

Interestingly, while the tendency to prune “cold” structures persists, the pruning curves vary depending on the initial dimensionality of the input features. In Cora, which has high-dimensional input features, the performance difference between pruning “hot” and “cold” embedding dimensions is less pronounced compared to input feature pruning, especially under the SGD optimizer. Pruning “hot” embedding dimensions has a smaller impact on performance compared to pruning “hot” input features. This is likely due to the embedding space having significantly lower dimensionality than the input space. The effect becomes even more evident in deeper layers: when examining the second GNN layer instead of the first one (see Fig. 11), the performance gap between the pruning curves narrows further. We attribute this behavior to the progressive reduction in temperature observed in deeper layers of the network. On the other side, for Pubmed, which has a lower input dimensionality than Cora, pruning “hot” embedding dimensions leads to a more noticeable performance drop with respect to pruning “cold” embedding dimensions, as in the case of pruning input features.

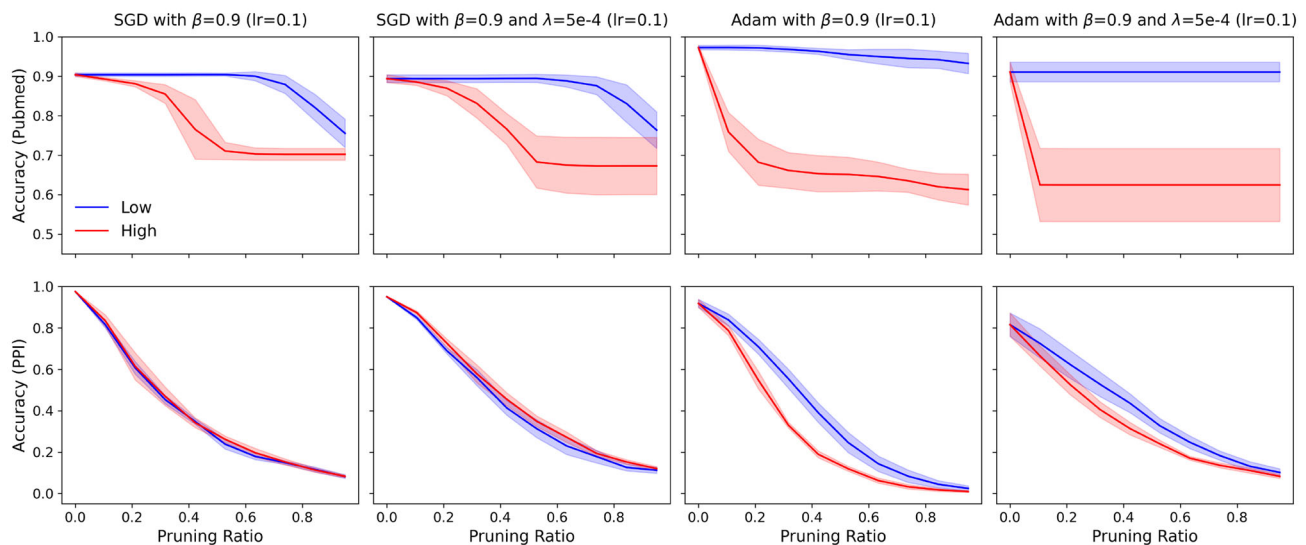


Fig. 12 Performance after zeroing-out embedding dimensions in the first GNN layer and re-evaluating the model on the pruned test set based on SGD-temperature (Pubmed and PPI datasets)

Fig. 13 CNN architecture for image classification on MNIST

CNN Architecture:

- Conv2d(input $\rightarrow$ 12, kernel=5)
- ReLU
- Conv2d(12 $\rightarrow$ 12, kernel=3)
- ReLU
- MaxPool2d(2)
- Dropout(0.25)
- Conv2d(12 $\rightarrow$ 24, kernel=5)
- ReLU
- Conv2d(24 $\rightarrow$ 24, kernel=3)
- ReLU
- MaxPool2d(2)
- Dropout(0.25)
- Linear(24 * 2 * 2 $\rightarrow$ 10)

Furthermore, in PPI, where the input dimensionality is even smaller and the embedding dimension exceeds the input dimension, pruning “cold” embedding dimensions in PPI performs even worse than pruning “cold” input features.

D Comparison with pruning filters

We conduct a comparative analysis of the three pruning criteria (temperature, SGD-temperature, and magnitude), when pruning filters from the CNN architecture shown in Fig. 13. This architecture is the same we employed in our previous work on filter pruning [1] and we train it on MNIST dataset [52]. We experiment with both SGD and Adam optimizers, setting the learning rate to 0.1. For SGD, we explore configurations with momentum ($\beta = 0.9$) and with or without L_2 weight decay ($\lambda = 5 \times 10^{-4}$). For Adam, we employ the standard first and second moment estimates ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) and consider setups with or without L_2 weight decay ($\lambda = 5 \times 10^{-4}$). Training is performed using cross-entropy loss over the 10 classes, with the number of equilibrium epochs set to 20. Training is repeated 10 times with a fixed random seed.

In Figs. 14 and 15, we show the pruning curves obtained when zeroing out filter weights for an increasing number of filters, going from the “hottest” to the “coldest” and vice versa. After pruning, we evaluate again the pruned model on the test set and we look at how the performance has changed with respect to the original setting. Figure 14 corresponds to pruning the first layer, while Fig. 15 to the second one.

The magnitude criterion, measuring the absolute value of filter weights, exhibits consistent behavior across optimizers: pruning filters with the lowest magnitude yields the best results, in agreement with the literature [26]. On the other side, when using temperature as a pruning criterion, we again observe a divergence between optimizers: for SGD, pruning filters with the lowest kinetic energy (“cold” filters) is optimal, whereas for Adam, pruning filters with the highest kinetic energy (“hot” filters) performs

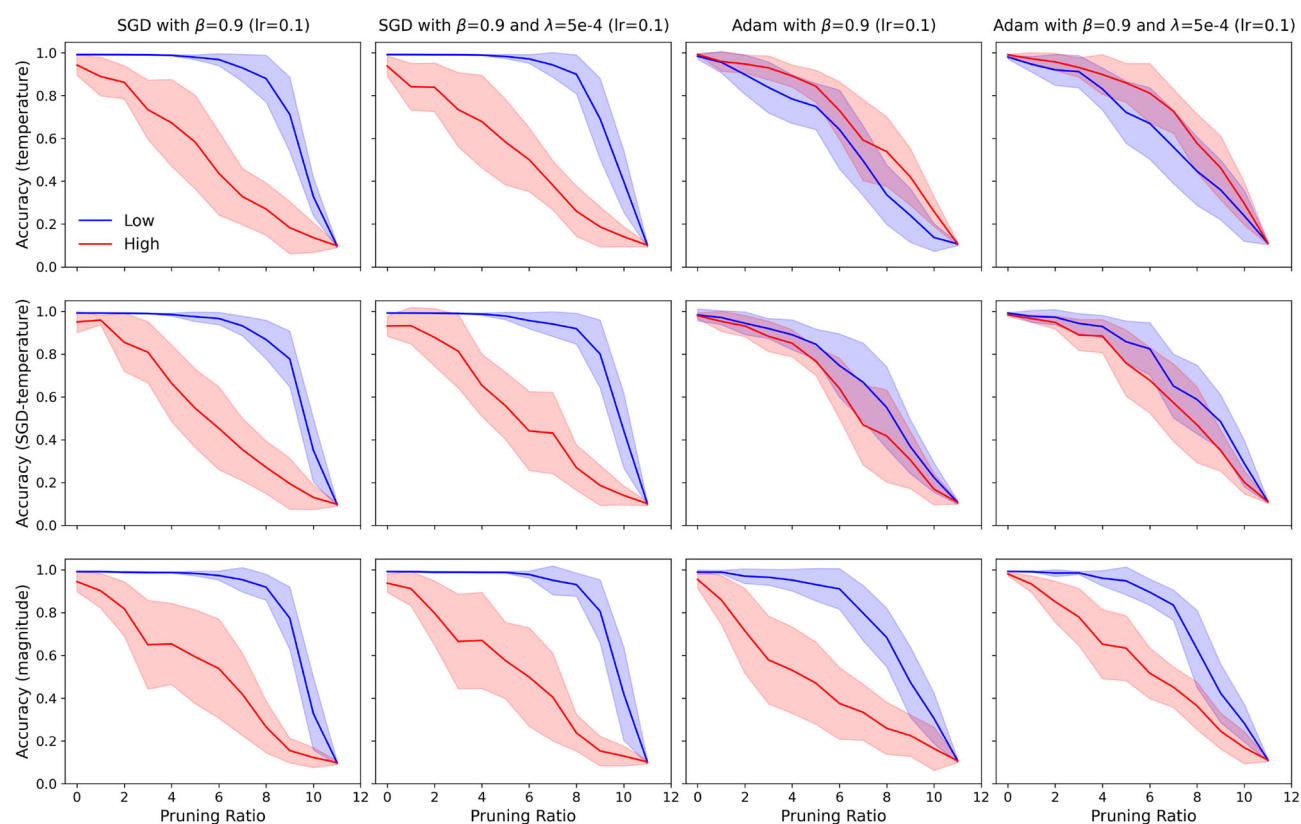


Fig. 14 Performance after zeroing-out filters in the first CNN layer and re-evaluating the pruned model on the test set (MNIST dataset)

better. This observation aligns with our earlier findings on feature pruning in GNNs in Sec. 4. Finally, the SGD-temperature criterion shows consistent behavior across optimizers, indicating that pruning filters with low gradient activity leads to the best performance. As noted in the previous sections, the performance difference between pruning strategies diminishes in deeper layers, with the gap between pruning curves narrowing significantly when moving from the first (Fig. 14) to the second layer (Fig. 15).

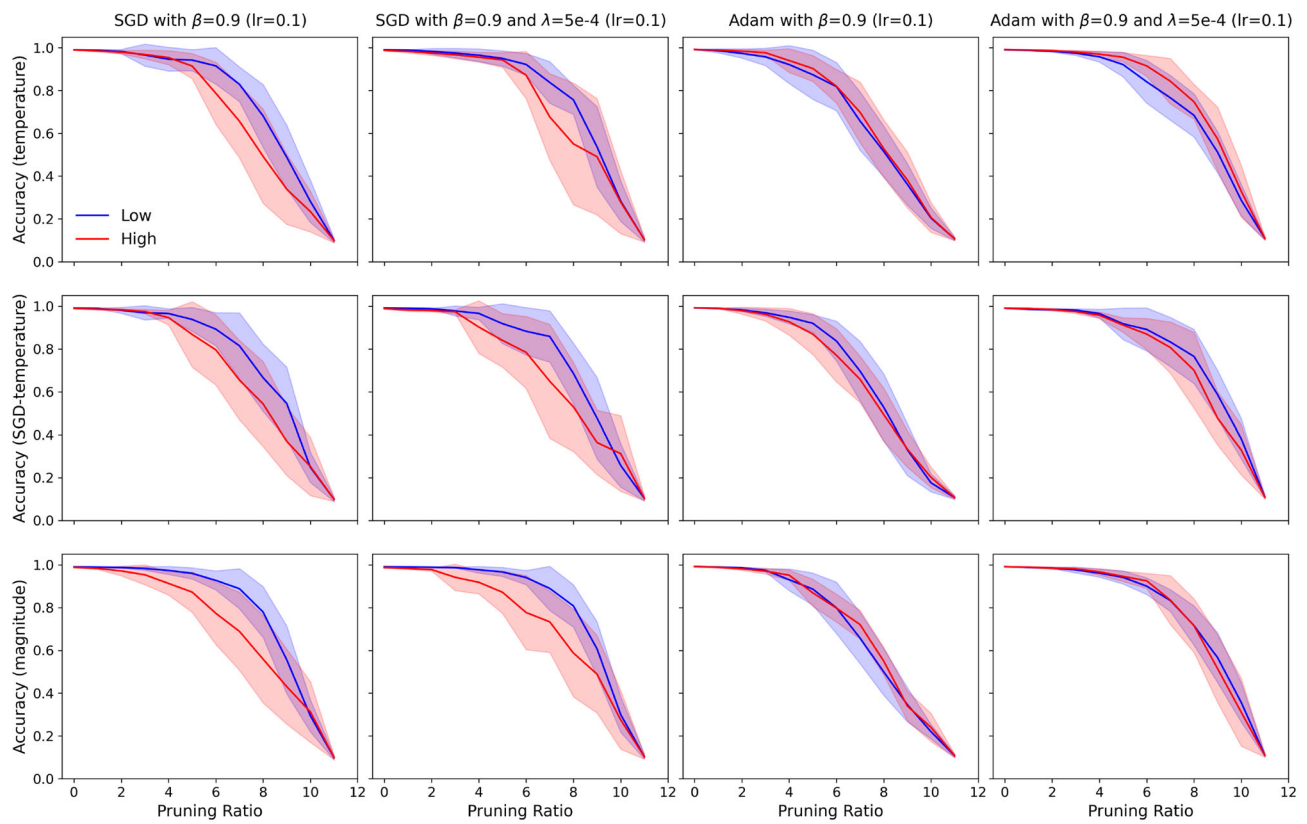


Fig. 15 Performance after zeroing-out filters in the second CNN layer and re-evaluating the pruned model on the test set (MNIST dataset)

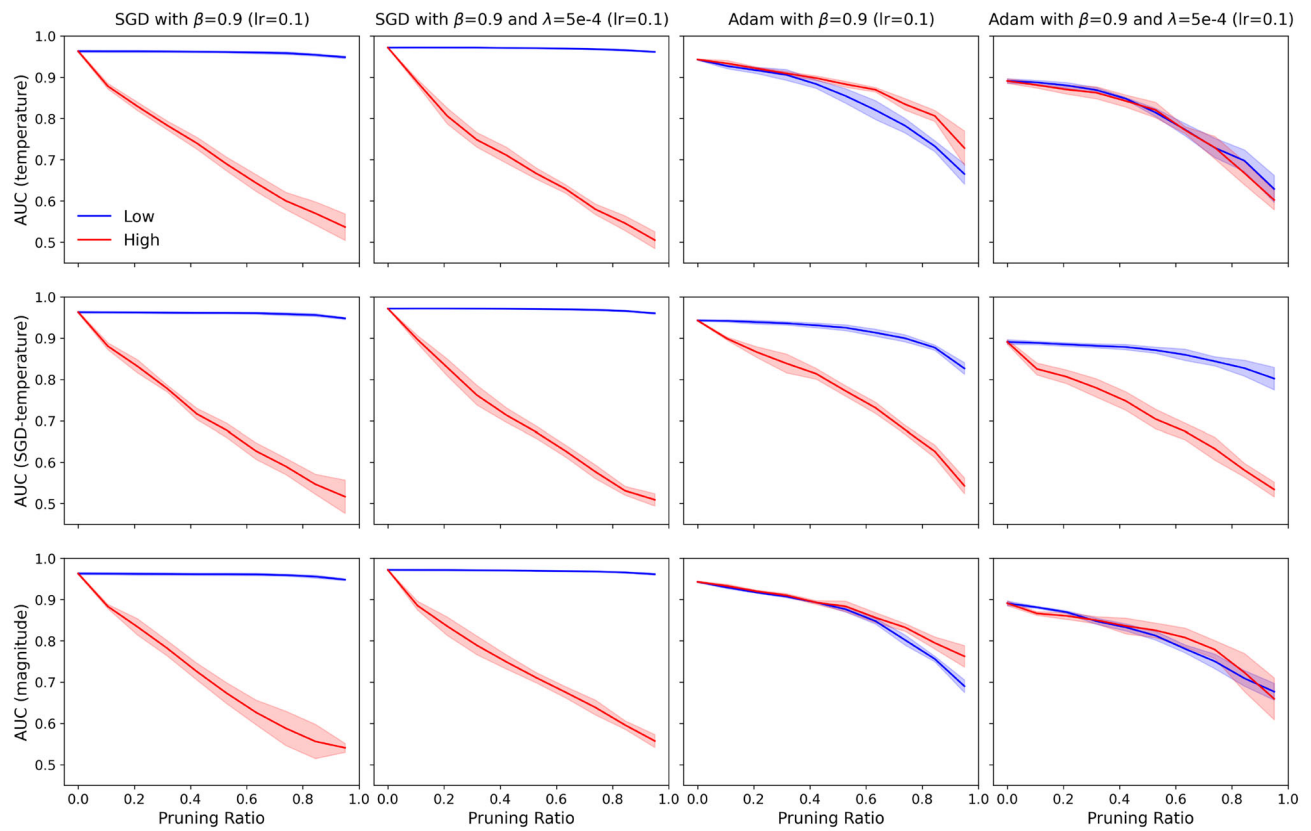


Fig. 16 Performance after zeroing-out features and re-evaluating the model on the pruned test set based on temperature (Citeseer dataset)

E Pruning curves on Citeseer

In Fig. 16, we present the pruning curves for the Citeseer dataset, which are obtained by progressively zeroing out features based on temperature, SGD-temperature, and magnitude. Pruning based on magnitude and temperature yields inconsistent results across optimizers as shown in Sec. 4. In contrast, SGD-temperature provides a clear behavior, consistently identifying low SGD-temperature features as the ones to prune. Furthermore, we observe that both temperature and magnitude align well with SGD-temperature when using the SGD optimizer. This behavior is also evident in Fig. 3, for the Cora dataset. However, this correspondence breaks down when using the Adam optimizer.

References

1. M. Lapenna, F. Faglioni, R. Fioresi, Thermodynamics modeling of deep learning systems for a temperature based filter pruning technique. *Front. Phys. Sec. Statist. Comput. Phys.* (2023). <https://doi.org/10.3389/fphy.2023.1145156>
2. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. *Nature* **521**(7553), 436–444 (2015)
3. W.J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, B. Yu *Interpretable machine learning: definitions, methods, and applications*. Proceedings of the National Academy of Sciences, PNAS, <https://doi.org/10.1073/pnas.1900654116>, (2019)
4. Vaishak Belle, Ioannis Papantonis, Principles and Practice of Explainable Machine Learning. *Frontiers in Big Data. Sec. Data Mining and Management* 4–2021 (2021). <https://doi.org/10.3389/fdata.2021.688969>
5. E.T. Jaynes, Information theory and statistical mechanics. *Phys. Rev.* **106**(4), 620 (1957)
6. J. Hopfield, J. Neurons, With graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences, PNAS* **81**(10), 3088–3092 (1984)
7. D.H. Ackley, G.E. Hinton, T.J. Sejnowski, A learning algorithm for boltzmann machines. *Cogn. Sci.* **9**(1), 147–169 (1985)
8. R. Salakhutdinov, G. Hinton (2009) *Deep Boltzmann machines*. Twelfth International Conference on Artificial intelligence and statistics, 440–455. PMLR,
9. R. Veiga, L. Stephan, B. Loureiro, F. Krzakala, L. Zdeborová *Phase diagram of Stochastic Gradient Descent in high-dimensional two-layer neural networks*. 36th Conference on Neural Information Processing Systems, NeurIPS, (2022)
10. D. Zhang *On the Temperature of Machine Learning Systems*. <https://api.semanticscholar.org/CorpusID:269292921>, (2024)
11. P. Chaudhari, S. Soatto *Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks*. Information Theory and Applications Workshop (ITA), pages 1–10, <https://doi.org/10.1109/ITA.2018.8503224>, (2018)
12. P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun et al., Entropy-sgd: biasing gradient descent into wide valleys. *J. Stat. Mech: Theory Exp.* (2019). <https://doi.org/10.1088/1742-5468/ab39d9>
13. R. Fioresi, P. Chaudhari, S. Soatto, A geometric interpretation of stochastic gradient descent using diffusion metrics. *Entropy* (2020). <https://doi.org/10.3390/e22010101>
14. M. Welling, Y.W. Teh. *Bayesian learning via stochastic gradient langevin dynamics*. ICML'11: Proceedings of the 28th International Conference on Machine Learning, <https://doi.org/10.5555/3104482>, (2011)
15. F. Barbaresco, Lie group statistics and lie group machine learning based on Souriau lie groups thermodynamics and koszul-souriau-fisher metric: new entropy definition as generalized casimir invariant function in coadjoint representation. *Entropy* **22**(6), 642 (2020)
16. C.-M. Marle (2016) *From tools in symplectic and Poisson geometry to J. M. Souriau's theories of statistical mechanics and thermodynamics*. *Entropy*, 18(10):370,
17. A. Anahory Simoes, M. De Leon, M. Lainz Valcazar, D.M. De Diego. *Contact geometry for simple thermodynamical systems with friction*. *Proceedings of the Royal Society A*, 476(2241):20200244, (2020)
18. R. Fioresi, F. Faglioni, F. Morri, L. Squadrani. *On the thermodynamic interpretation of deep learning systems*. *Geometric Science of Information: 5th International Conference*, https://doi.org/10.1007/978-3-030-80209-7_97, (2021)
19. M. Lapenna, F. Faglioni, F. Zanchetta, R. Fioresi *Geometric Deep Learning: a Temperature Based Analysis of Graph Neural Networks.*, *Geometric Science of Information: 6th International Conference*, https://doi.org/10.1007/978-3-031-38299-4_65, (2023)
20. H. Yang, X. Lingao, Structured pruning for deep convolutional neural networks: a survey. *IEEE Trans. Pattern Anal. Mach. Intell.* (2024). <https://doi.org/10.1109/TPAMI.2023.3334614>
21. H. Cheng, M. Zhang, J.Q. Shi, A survey on deep neural network pruning: taxonomy, comparison, analysis, and recommendations. *IEEE Trans. Pattern Anal. Mach. Intell.* (2024). <https://doi.org/10.1109/TPAMI.2024.3447085>
22. J. Frankle, M. Carbin *The lottery ticket hypothesis: finding sparse, trainable neural networks*. *International Conference on Learning Representations, ICLR*, [arXiv:1803.03635](https://arxiv.org/abs/1803.03635), (2019)
23. M. Verbockhaven, S. Chevallier, G. Charpiat *Growing Tiny Networks: Spotting Expressivity Bottlenecks and Fixing Them Optimally.*, <https://arxiv.org/abs/2405.19816>, (2024)
24. H. Zhou, J.M. Alvarez, And fatih porikli less is more: towards compact cnns. *European Conference on Computer Vision* (2016). https://doi.org/10.1007/978-3-319-46493-0_40
25. W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks. *Adv. Neural Inf. Process. Syst. NIPS* (2016). <https://doi.org/10.5555/3157096.3157329>
26. H. Li, A. Kadav, I. Durdanovic, H. Samet, H. Peter Graf, Pruning filters for efficient convnets. *International Conference on Learning Representations ICLR* **1608**, 08710 (2017)
27. Y.L. Cun, J.S. Denker, S.A. Solli *Optimal Brain Damage*. *Advances in Neural Information Processing Systems 2, NIPS*, (1989)
28. B. Hassibi, D.G. Stork *Second order derivatives for network pruning: optimal brain surgeon*. *NIPS'92: Proceedings of the 5th International Conference on Neural Information Processing Systems*, (1992)
29. P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz. *Pruning Convolutional Neural Networks for Resource Efficient Inference*. *International Conference on Learning Representations ICLR*, (2017)
30. A.K. McCallum, K. Nigam, J. Rennie, K. Seymore, Automating the construction of internet portals with machine learning. *Inf. Retrieval* (2000). <https://doi.org/10.1023/A:1009953814988>
31. Giles C. Lee, K.D. Bollacker, S. Lawrence *CiteSeer: an automatic citation indexing system*. *Proceedings of the Third ACM Conference on Digital Libraries*, (1998)
32. W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec *Open Graph Benchmark: Datasets for Machine Learning on Graphs.*, 34th Conference on Neural Information Processing Systems (NeurIPS 2020), <https://arxiv.org/abs/2005.00687>, (2021)
33. P. Sen, G.M. Namata, M. Bilgic, L. Getoor, B. Gallagher, T. Eliassi-Rad *Collective Classification in Network Data*. *AI Magazine*, (2008)
34. Z. Marinka, A. Monu, L. Jure *Predicting multicellular function through multi-layer tissue networks*. *Bioinformatics*, (2017)
35. A. Deepak, Z. Huaming. *Feature Selection and Extraction for Graph Neural Networks*. *Proceedings of the 2020 ACM Southeast Conference*, (2020) <https://doi.org/10.1145/3374135.3385309>,
36. L. Chuang, M. Xueqi, Z. Yibing, D. Liang, D. Tao Dapeng, B.H. Wenbin, D.P. Mandic, Comprehensive Graph Gradual Pruning for Sparse Training in Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* (2024). <https://doi.org/10.1109/TNNLS.2023.3282049>
37. D.E. Rumelhart, G.E. Hinton, R.J. Williams *Learning representations by back-propagating errors*. *Nature*, (1986)
38. L.C. Yann, B. Léon, *Orr Genevieve B and Müller Klaus-Robert* (Tricks of the trade, Springer, Efficient BackProp. Neural networks, 2012)
39. Y. Nesterov. *A method for solving the convex programming problem with convergence rate $O(1/k^2)$* . *Proceedings of the USSR Academy of Sciences*, 269, 543–547, Available at: <https://api.semanticscholar.org/CorpusID:145918791> (1983)

40. D.P. Kingma, J.B. Ba *Adam: A method for stochastic optimization*. 3rd International Conference on Learning Representations, ICLR, (2015)
41. H. Risken, *The Fokker-Planck Equation* (Springer, Methods of Solution and Applications., 1996)
42. Q. Li, C. Tai, Weinan E (2017) *Stochastic modified equations and adaptive stochastic gradient algorithms*. Proceedings of the 34th International Conference on Machine Learning, 1511.06251,
43. Y.-A. Ma, T. Chen, E.B. Fox, A complete recipe for stochastic gradient mcmc. *Neural Inf. Proces. Syst* **1506**, 04696 (2015)
44. V. Petar, C. Guillem, C. Arantxa, R. Adriana, L. Pietro, B. Yoshua, Graph attention networks. *International Conference on Learning Representations* **1710**, 10903 (2018)
45. W.L. Hamilton, R. Ying, J. Leskovec *Inductive representation learning on large graphs*. Proceedings of the 31st International Conference on Neural Information Processing Systems, (2017)
46. J. Li, H. Shomer, H. Mao, S. Zeng, Y. Ma, N. Shah, J. Tang, D. Yin *Evaluating Graph Neural Networks for Link Prediction: Current Pitfalls and New Benchmarking*. Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS), Red Hook, NY, USA: Curran Associates Inc., pp. 169–182, (2023) Available at <https://github.com/Juanhui28/HeaRT>
47. S. Ratna, S. Singh, A. Sharma, *An inclusive analysis for performance and efficiency of graph neural network models for node classification*. *Computer Science Review*, 56, 100722, Available at: (2025) <https://doi.org/10.1016/j.cosrev.2024.100722>
48. S. Ioffe, C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:448–456, (2015)
49. R. Sato. *Graph neural networks can recover the hidden features solely from the graph structure*. Proceedings of the 40th International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, pp. 1248, JMLR.org, (2023)
50. M. Lapenna, C.D. Bacco. *How do Probabilistic Graphical Models and Graph Neural Networks Look at Network Data?* arXiv preprint [arXiv:2506.11869](https://arxiv.org/abs/2506.11869), . Available at: <https://arxiv.org/abs/2506.11869> (2025)
51. L. Balles, P. Hennig, *Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients*, Proceedings of the International Conference on Machine Learning (ICML), (2017). Available at: <https://api.semanticscholar.org/CorpusID:46884491>
52. L. Deng, The mnist database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* **29**(6), 141–142 (2012)