Dynamic Parameter Rank Pruning of a Singular Value Decomposed Multilayer Perceptron

Wander Stribos
w.h.stribos@student.utwente.nl
University of Twente
Enschede, The Netherlands

ABSTRACT

Neural networks are an extremely impressive technology, allowing computers to model the real world at a level of effectiveness not previously thought possible. However, more complex ones require massive amounts of computing power and storage, creating a limit to their potential. Thus, much research is done into finding ways to decrease the computational efficiency and memory usage of more complex models without losing functionality. One of these methods is the post-training compression technique 'singular value decomposition', which approximates a weight matrix with smaller matrices. To avoid the necessity of extensive retraining, the paper proposes a dynamic implementation of the same mathematical concept, which starts training with full-size decomposed matrices and prunes itself during training. Through batched pruning and a relaxed orthogonality constraint, the final system effectively decreases model size during training while increasing overall accuracy.

The full Tensorflow implementation is available at https://github.com/WanderStribos/CompleteImplementation.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; Batch learning; • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

multilayer perceptrons, neural networks, optimisation, singular value decomposition

ACM Reference Format:
Wander Stribos. 2018. Dynamic Parameter Rank Pruning of a Singular Value Decomposed Multilayer Perceptron. In *Proceedings of 43rd Twente Student Conference on IT (TScIT 42).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Neural networks are a remarkable technology, with applications ranging from computer vision to natural language processing. [7] However, they can easily grow to bizarre sizes, with larger systems using millions or sometimes even billions of parameters. This high level of computational workload and memory creates a limit to their efficacy, especially on embedded systems with stronger memory and computational constraints. [5] Because of this, much research is done on methods to make models that have similar levels of efficacy as larger systems whilst requiring less computational power, such as pruning connections with minimal effect on the total calculations from the system [5], quantisation [10], or a multitude of novel techniques to filter out parts of the input before training the model. [11] [8]

While there are many of these techniques, the effective compression of neural networks remains a significant open problem; a quick search of "neural network compression" on DBLP returns sixteen published scientific articles and four conference papers in the first four months of 2025 alone. One of these techniques is truncated singular value decomposition (SVD), [4], a post-training compression method which has been shown to be effective in neural networks. [2] However, it often requires significant re-training as the approximated matrix often performs at a significantly lower level. [6] Because of this, there have been attempts at implementing this technique in a dynamic fashion, wherein the rank is decreased in-training. In Chung et al [2019], a sparsity constraint is applied to weight matrices initialised in truncated SVD form, which is otherwise trained normally. [3] In Sharma et al [2025], the SVD structure was enforced by means of a structural loss function throughout training. This allowed the system to truncate the weights during training dynamically by only checking the singular values. However, the paper contained an odd inconsistency: The compression loss, the part of the system that pushes the lower ends of the model to close-to-zero values before pruning, as described, only affects values that have already been pruned, which would, thus, no longer be part of the calculation. While it could be interpreted as the model simply compressing 'pruned' values to zero instead of actively removing them, the paper clearly mentions removing the parameters mid-training. [9]

In this paper, a novel approach is proposed that implements the pruning in batched form, where both compression and pruning are applied to multiple ranks simultaneously. To allow the model to handle the increase in effect from pruning in this manner, the orthonormality constraint of column vectors is replaced with a mathematically simpler normalisation constraint. This significantly increases the model's flexibility in handling its decreasing size, considerably reducing the decrease in accuracy that previously accompanied pruning. Applying this technique to a simple multilayer perceptron both significantly compressed the model's size and increased its accuracy over the control model.

TScIT 42, July 4, 2025, 2018, Enschede, The Netherlands

2 BACKGROUND

2.1 Multilayer perceptron

Multilayer perceptrons are a type of neural network that consists of one or more fully connected layers, each with an activation function. The fully connected layer consists of a weight matrix $W \in \mathbb{R}^{m \times n}$ and a bias layer $b \in \mathbb{R}^n$. During the forward pass, it takes an input vector of size m and transforms it into an output vector of size n through

$$Z = XW + b. \tag{1}$$

[2] Afterwards, a continuous activation function such as $ReLU(x) = max(0, x)$ or $sigmoid(x) = \frac{1}{1+e^{-x}}$ is applied, after which the output is fed into the input of the new layer. After one or more hidden layers, they reach the output layer. This is also a fully-connected layer, which attempts to transform the outputs of the previous layer(s) into an interpretable output, such as a scalar for a regression task or a one-hot encoded output for a classification task. [1] They are one of the first forms of what later became deep neural networks.

2.2 Singular Value Decomposition

Given a matrix $W \in \mathbb{R}^{m \times n}$, a singular value decomposition (SVD) of W are orthonormal matrices $U \in \mathbb{R}^{m \times r}$, $V \in \mathbb{N}^{n \times r}$ and diagonal matrix $\Sigma \in \mathbb{R}^{r \times r}$ consisting of positive singular values $\boldsymbol{\sigma} = \{\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2, ..., \boldsymbol{\sigma}_r\} = \{\Sigma_{1,1}, \Sigma_{2,2}, .., \Sigma_{r,r}, \}$ such that $W = U\Sigma V^T$, where $r \leq min(m, n)$ is the rank of the matrix. The SVD of a matrix does not have to be unique, but there always exists a unique decomposition such that $\boldsymbol{\sigma}_1 \geq \boldsymbol{\sigma}_2 \geq ... \geq \boldsymbol{\sigma}_r > 0$. Since the later, smaller singular values have less effect on the recomposition, it is possible to truncate the matrices into a low-rank approximation $U_t \in \mathbb{R}^{m \times k}, V_t \in \mathbb{R}^{n \times k}, \Sigma_t \in \mathbb{R}^{k \times k}$, where the new rank $k \leq r$ and $U_t \Sigma_t V_t^* = \widetilde{W} \approx W$. With k sufficiently small, one can store $\widetilde{W}$ as two matrices $U\Sigma$ and V^T, which consist of $k(m + n)$ numbers, instead of the original $m \times n$ numbers. [4] This technique can be straightforwardly applied to fully-connected layers in a neural network, transforming the classical formula in Equation 1 into

$$Z = (XU\Sigma)V^T + b$$

as described in [2].

3 PROPOSED METHOD

3.1 Loss function

During training, the model uses a simple stochastic gradient descent to minimise a triple loss function. With D being the list of dense layers where the dynamic technique is applied, it is defined as

$$L_{total} = L_{app} + \sum_{d \in D} L_{struct}(d) + L_{comp}(d), \tag{2}$$

which is mostly based on the function described in [9]. However, in this paper, two novel techniques are implemented through a reimagining of the L_{comp} and L_{struct} functions. L_{app}, the application loss, is calculated in the same way as it would in a normal scenario. (In the experiment, categorical cross-entropy is used.)

3.1.1 Compression loss. L_{comp} paves the way for the dynamic pruning by pushing the last θ singular values to zero with hyperparameters μ_{comp} and θ, the compression count:

$$L_{comp} = \mu_{comp} \times \sum_{i=k-\theta}^{k} \boldsymbol{\sigma}_i \tag{3}$$

3.1.2 Original structural loss. The structural loss function ensures that the SVD structure of the layer remains intact throughout the training process. Originally, this was implemented in a very similar way to [9] and is, itself, also divided into subfunctions, with μ_{ort} and μ_{sing} being hyperparameters that can be adjusted per layer:

$$L_{struct} = \mu_{ort} L_{ort} + \mu_{sing}(L_{negs} + L_{sort}). \tag{4}$$

L_{ort} is tasked with retaining the orthonormality of the U and V matrices. By ensuring the column vectors are of equal length, the sizes of the singular values become the primary metric for gauging the effects that decreasing the rank has on the total calculation. It is defined as follows, with ϕ being an assisting function that takes the average of all values, squared in a matrix:

$$L_{ort} = \phi(U_t^T U_t - I_{k \times k}) + \phi(V_t^T V_t - I_{k \times k}). \tag{5}$$

L_{negs} is tasked with keeping the singular values positive. To prevent layers with a large rank from dominating the calculation, it takes the average of all negative singular values. It returns zero if there are none, which is omitted from the formula for readability:

$$L_{negs} = \frac{\{max(-\boldsymbol{\sigma}_i, 0) | i \in \{1..k\}\}}{\text{\#non-zero values in the above set}}. \tag{6}$$

L_{sort} is tasked with keeping the singular values sorted from large to small. It is similar to L_{negs}, but calculates the average of all positive differences of values with the one before. Once again, it returns zero if there is no average to calculate:

$$L_{sort} = \frac{\{max(\boldsymbol{\sigma}_{i+1} - \boldsymbol{\sigma}_i, 0) | i \in \{1..k-1\}\}}{\text{\#non-zero values in the above set}} \tag{7}$$

With careful balancing of hyperparameters, L_{struct} pushes the model to only learn through rotations of the U and V matrices and scaling of Σ.

3.1.3 Improved structural loss. The previous method did not yield the desired results. However, the author noticed a few possible points of improvement, and a new structural loss function was developed:

$$L_{struct} = \mu_{norm} L_{norm} + \mu_{sing}(L_{sort} + max(0, -\sigma_k)). \tag{8}$$

In this formula, L_{negs} was replaced with a significantly less mathematically complex calculation that only kept the final value positive, as L_{sort} already ensured the other values were larger and therefore positive as well. Additionally, L_{ort} was replaced with L_{norm}, defined as

$$L_{norm} = mean\Big(\sum_{\boldsymbol{v} \in U_t, V_t} |1 - \sum_{x \in \boldsymbol{v}} x^2|\Big) \tag{9}$$

where $\boldsymbol{v}$ are the column vectors comprising U and V. Through this simplified norm calculation, the columns are still pushed to

remain unit vectors. Meanwhile, the removal of the orthogonality constraint grants the system the flexibility to compensate for vectors under compression by rotating the remaining vectors to decrease their distance from the to-be-pruned vector set.

3.2 Dynamic pruning

Through L_{comp}, the last θ values in the singular values $\boldsymbol{\sigma}_t$ are pushed to zero. Since they will have less effect on the overall calculation, they can be pruned without too significant impact. During training, the system monitors the ratio between the last θ and second-to-last θ values, and decreases the rank k by θ if this ratio is more than ϵ, the pruning threshold hyperparameter. In other words, if:

$$\sum_{i=k-\theta}^{k} \boldsymbol{\sigma}_i < \epsilon \sum_{j=k-(2\times\theta)}^{k-\theta} \boldsymbol{\sigma}_j. \tag{10}$$

θ and ϵ need to be carefully set: the higher their values, the faster the system can compress. On the other hand, it also results in larger parts of the system being pruned at once, causing temporary drops in accuracy that require the system to readjust. For θ, a size of around 1-2% of the starting rank appeared to be optimal; much lower values would result in the system converging long before enough compression was reached, requiring artificially high training times just for compression, while higher values would result in too high drops in accuracy.

3.3 Recompiling and fine-tuning

With k being sufficiently low, saving $\widetilde{W}$ as $U_t\Sigma_t$ and V_t^T can take up significantly less storage space. However, a non-truncated singular-value decomposed matrix (or simply, one where k remains the starting value) saved in this manner can take up up to twice the size of the original matrix. To ensure the parameter count of each dynamic layer is at worst the size of a non-decomposed layer with the same total effect on the input, the model recompiles itself near the end of training. During this recompiling, it checks each layer to determine whether keeping the weight matrix decomposed saves storage space. If it doesn't, it 'recomposes' the matrix and saves the matrix as singular $\widetilde{W}$.

Afterwards, the system fine-tunes for a few epochs while only minimising the application loss. This has two reasons:

- As mentioned before, pruning is often paired with a temporary drop in accuracy. By abstaining from pruning for these final few epochs, the model is prevented from stopping in the middle of one of these dips.
- Additionally, during this last bit of fine-tuning, the model can fully prioritise accuracy without focusing on structural and compression rules. This results in a final (albeit often minor) increase in its accuracy.

4 EXPERIMENTAL SETUP

4.1 Overal setup

To allow for enough experimentation to study the effects of various hyperparameters, a relatively simple setup was used: A three-layer multilayer perceptron with ReLu activation functions after the hidden layers and a SoftMax function after the final one, as shown in Figure 1. This model was then trained on the MNIST dataset, which consists of 28×28-pixel images of written digits, the goal being recognising which digit was written. The original model was trained for 200 epochs, followed by 10 fine-tuning epochs. However, the new version converged significantly faster and trained for 150 epochs before the 10 fine-tuning epochs.

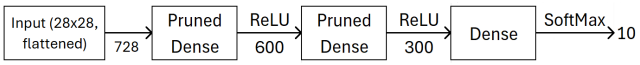

Figure 1: The model architecture used to test the dynamic layers. Between each dense layer, the activation function (top) and vector size (bottom) are specified.

The output layer was not included in the pruning, as decreasing its rank could quickly result in it no longer being able to classify ten different outputs, logically causing a significant drop in accuracy.

4.2 Original comparison

4.2.1 Control model. To establish a baseline accuracy, the model was compared with a non-SVD model of the same structure, which did not attempt the pruning and ran for 150 epochs, followed by 60 epochs with a lower learning rate to allow it to fine-tune.

4.2.2 Traditional SVD. Additionally, the model was compared to a more traditional SVD setup. In this form, it ran for a mere 150 epochs, after which SVD is applied to each layer's weights. Then, with hyperparameter pruning threshold ϵ, it finds largest k for which $\boldsymbol{\sigma}_k > \epsilon\boldsymbol{\sigma}_1$ and truncates everything after k. Similar to the dynamic system, it then recompiles, with its layers either remaining as-is or being saved as SVD layers (depending on which is more memory-efficient), and trains for a final 60 epochs, once again with a lower learning rate. The reasoning behind the fine-tuning taking up a much larger part of the training is simple. Since the pruning occurs all at once instead of in smaller increments, it results in a significantly higher accuracy drop, which it must recover from.

4.3 Comparison improved version.

Since the improved version was trained for fewer epochs, it required comparisons that also had these constraints. In these versions, the first training consisted of 110 epochs, followed by 50 fine-tuning epochs. Additionally, a 'non-pruning' version of the dynamic model was trained, which had the same setup as the dynamic model but with μ_{comp} set to 0. The smaller control models were trained for 130 and 40 epochs.

4.4 Results

The results for the original implementation can be found in Table 1. It consists of the control model, three different degrees of compression for the dynamic method, and two traditional SVD models with different pruning thresholds. Additionally, it contains two standard models, which were initialised to have a similar parameter count as the most effective SVD method. The full configuration of the systems can be found in the appendix.

The results were clear: the original dynamic method was not an improvement over traditional singular value decomposition as a post-hoc method, or simply training a simpler model in the

Table 1: Accuracies and parameter counts for the first dynamic implementation. While it successfully decreased the model size, this came with a higher decrease in accuracy compared to traditional post-hoc SVD with retraining.

Compression method	Parameter count start	Parameter count final (size diff.)	Test set accuracy
None (control)	654.316	654.316 (-0%)	97.47%
Dynamic (weak)	1.105.222	461.058 (-29.54%)	95.30%
Dynamic (medium)	1.105.222	372.246 (-43.11%)	93.31%
Dynamic (strong)	1.105.222	287.118 (-56.12%)	83.39%
Static SVD (weak)	654.316	445.493 (-31.91%)	97.25%
Static SVD (strong)	654.316	120.538 (-81.58%)	96.33%
None (140, 70)	120.486	120.486 (-81.59%)	94.05%
None (100, 400)	122.916	122.916 (-81.21%)	97.19%

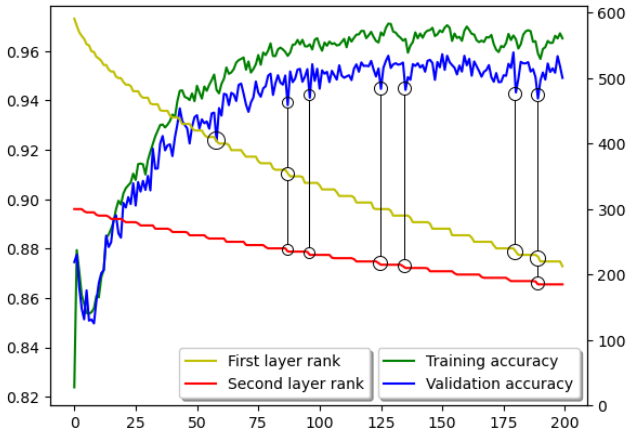

Figure 2: Accuracy (left axis) and rank (right axis) during the first epochs of the original dynamic model with weak compression. One can see how the rank decreases very quickly at first, yet slows down over the course of training as the system gets more efficient with its parameter usage. It also reveals the weakness of the original system: by not being able to compensate in the direction of compressed vectors through rotation of the remaining vectors, accuracy often drops after pruning, resulting in a decrease in final accuracy. Some of the more clear accuracy dips due to pruning are indicated.

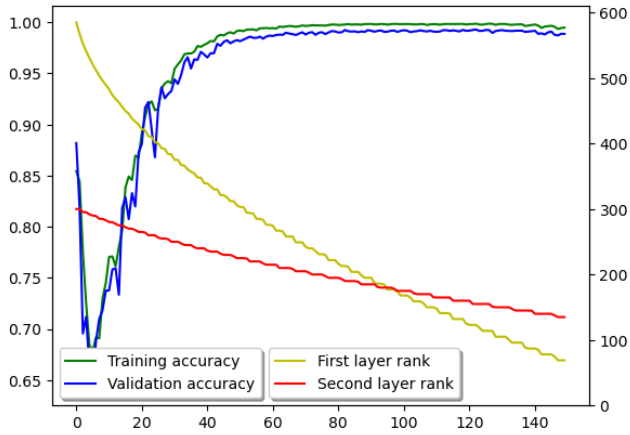

Figure 3: Accuracy (left axis) and rank (right axis) during the first 150 epochs of the improved model at strong settings. While a few smaller pruning dips are visible in a few of the final epochs, they are significantly less present than during the original setup.

first place. While it successfully decreased its rank dynamically during training, it did so at a high cost in accuracy compared to the traditional method. Additionally, the training in decomposed form meant that the model started training with almost twice the amount of parameters, and the triple loss function gave it a significantly higher computational load per parameter.

In Figure 2, the rank and accuracy of the weak dynamic model are shown. The plot reveals how the rank decreases very quickly at first, yet slows down over the course of training once the system gets more efficient with its parameter usage. One can also see how pruning occasionally results in minor accuracy drops, which the system usually recovers from within a few epochs.

However, the results for the improved setup, which are presented in Table 2, tell a much more positive tale. Even with a 24% reduction in training epochs, the dynamic models outclass even the control groups from the previous experiment. This appears to be in part due to the increased complexity of the model, as the dynamic setup with no compression also gave a noticeably higher result than the control group. However, the self-pruning models still managed to get the highest result. This can be explained by taking a closer look at the strong dynamic model and the non-pruning model. While the training accuracy for the non-pruning model was slightly higher than the training accuracy for the strong model, the latter still had a higher validation accuracy. This implies the pruning to work as a form of regularisation, guarding against overfitting.

The graphs do not show the final fine-tuning step. However, the final increase from this step is often not too large, except for some early experiments with far too strong pruning that happened to prune right before recompilation.

Figure 5 contains an example graph for the results of a too-aggressive configuration. It effectively highlights one of the system's main limitations: the inability to determine when to stop

Table 2: Accuracies and parameter counts for the improved setup, including the percentual difference in size to the control model. Our dynamic method both increases accuracy and significantly decreases model size.

Compression method	Param. count start	Param. count final (size diff.)	Training set acc.	Test set acc.
None (control)	654.316	654.316 (-0%)	97.10%	95.07%
Dynamic (weak)	1.105.222	498.758 (-23.77%)	99.96%	**99.42%**
Dynamic (medium)	1.105.222	397.718 (-39.22%)	99.98%	99.36%
Dynamic (strong)	1.105.222	220.914 (-66.24%)	99.74%	98.98%
Dynamic (none)	1.105.222	654.316 (-0%)	99.78%	98.81%
Static SVD (weak)	654.316	422.782 (-35.39%)	97.29%	95.48%
Static SVD (strong)	654.316	**139.906 (-78.62%)**	95.81%	94.18%
None (140, 70)	120.486	**120.486 (-81.59%)**	97.31%	96.76%
None (100, 400)	122.916	122.916 (-81.21%)	98.21%	94.05%

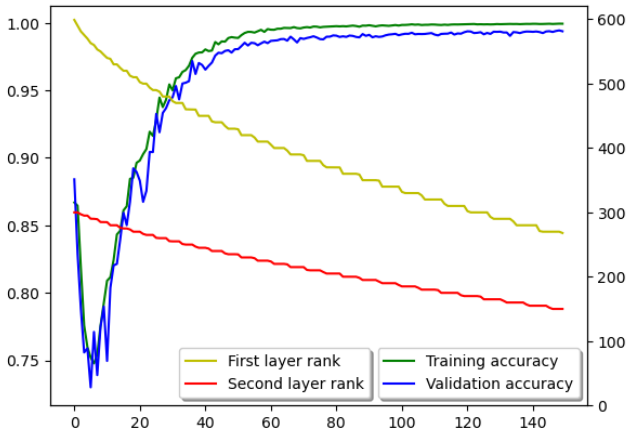

Figure 4: Accuracy (left axis) and rank (right axis) during the first 150 epochs of the improved model at weak settings. This version only pruned 23.77% of the original size. However, it boasted the highest testing accuracy at 99.42%.

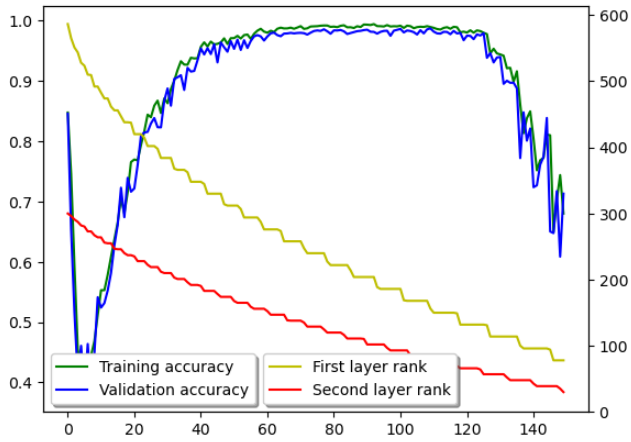

Figure 5: An example of the dangers of over-enthusiastic compression. In this setup, compression batch sizes θ were set to 3% of the ranks, and pruning threshold ϵ was set to 0.7 and 0.6. Although the improved version has significantly smaller accuracy drops from pruning, the accuracy still dropped from 90% to 70% over the final 25 epochs (including fine-tuning epochs).

pruning. If configured to prune too intensely, the system will simplify itself to the point where it can no longer accurately represent reality. On the other hand, if the model does not adequately prioritise compression, it will not compress itself sufficiently and recompile itself to full size.

5 LIMITATIONS AND FUTURE WORK

The most obvious limitation of this work is the lack of environments in which the technique has been tested. Once the system was set up, the final version was only tested on a simple three-layer MLP using a relatively simple dataset due to time constraints. While some testing was conducted on different output sizes of hidden layers, it was not documented extensively enough, nor was it compared to different setups. Additionally, the improved models all had accuracies of ~ 99%. This made it difficult to compare results, as even a tenth of a percent difference in accuracy would be significant. More research should be conducted using more complex model architectures, as the technique can be applied to any model architecture with fully-connected layers. This would also allow for more up-to-date comparisons, as the models were compared to relatively simple control groups instead of state-of-the-art models.

Additionally, the ablation study in Table 3 revealed that the L_{sing} function had surprisingly little effect on the model. An inspection of the contents of $\boldsymbol{\sigma}_t$ showed that the values were still sorted and positive at the end of the training. Also, except for the values that were under compression during the last few epochs, they were almost exactly the same as on initialisation. This was initially thought to be an unexpected consequence of removing the orthogonality constraint. However, a quick experiment in which L_{sing} was disabled in the original models showed similar results for $\boldsymbol{\sigma}_t$. Instead, it could be explained by the differences in sizes to the weights in decomposed form; While $\boldsymbol{\sigma}$ could contain values of above 50, the column vectors of U and V were all unit vectors with average values that often floated around 0.03 and 0.06 in the first and second layer,

Table 3: Ablation study on the various parts of the loss function, done on the improved dynamic model on medium strength. It reveals that L_{sing} has surprisingly little effect on the results.

Disabled subfunction	Param. count final (size diff.)	Training set acc.	Test set acc.
None (control)	397.718 (-39.22%)	99.98%	99.36%
L_{norm}	245.478 (-62.48%)	59.05%	57.35%
L_{sing}	245.478 (-62.48%)	99.94%	99.27%
L_{struct}	227.138 (-65.29%)	61.18%	61.27%
L_{comp}	654.316 (-0%)	99.43%	98.34%

respectively. This meant that, with the optimiser having a maximum amount it could change each weight through norm clipping, vector rotation simply had a larger relative effect on the forward pass compared to attempting to slowly update the large singular values. However, further research is needed to determine whether the loss function remains unnecessary in more rigorously trained models, as the differences from the initial weights could become significantly larger.

The technique could have also been significantly improved by implementing safeguards against over-pruning. As shown in Figure 5, the system will continue pruning even when it is still attempting to recover from previous pruning. In a future implementation, the threshold ϵ and batch size θ could be set to decrease over time, E.G. by having θ be relative to the current rank, having both parameters decrease depending on the accuracy drop from the last prune, or pruning could be temporarily paused until the accuracy recovered enough from the last prune.

6 CONCLUSION

In this work, we describe and implement an in-training compression technique inspired by singular value decomposition for fully-connected layers in neural networks. We demonstrate how the technique functions substantially better without the orthogonality constraint. This results in a method that, given that the hyperparameters are properly set, dynamically decreases the model's parameter count during training whilst boasting significantly increased accuracy over models trained normally.

7 APPENDICES

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Matan Ben Noach and Yoav Goldberg. 2020. Compressing Pre-trained Language Models by Matrix Decomposition. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, Kam-Fai Wong, Kevin Knight, and Hua Wu (Eds.). Association for Computational Linguistics, Suzhou, China, 884–889. https://doi.org/10.18653/v1/2020.aacl-main.88

[3] Hoon Chung, Euisok Chung, Jeon Gue Park, and Ho-Young Jung. 2019. Parameter Reduction For Deep Neural Network Based Acoustic Models Using Sparsity Regularized Factorization Neurons. In *2019 International Joint Conference on Neural Networks (IJCNN)*. 1–5. https://doi.org/10.1109/IJCNN.2019.8852021

[4] Reinsch C. Golub, G.H. 1970. Handbook Series Linear Algebra. Singular Value Decomposition and Least Squares Solutions. *Numer. Math.* 14 (1970), 403–420. http://eudml.org/doc/131963

[5] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf

[6] Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. In *International Conference on Learning Representation*.

[7] Y. LeCun, Y. Bengio, and G. Hinton. 2015. Deep learning. *Nature* 521 (2015), 436–444. https://doi.org/10.1038/nature14539

[8] Mohamed Saied, Shawkat Guirguis, and Magda Madbouly. 2025. Review of filtering based feature selection for Botnet detection in the Internet of Things. *Artificial Intelligence Review* 58 (2025). https://doi.org/10.1007/s10462-025-11113-0

[9] Manish Sharma, Jamison Heard, Eli Saber, and Panagiotis Markopoulos. 2025. Convolutional Neural Network Compression via Dynamic Parameter Rank Pruning. *IEEE Access* 13 (2025), 18441–18456. https://doi.org/10.1109/ACCESS.2025.3533419

[10] Vincent Vanhoucke, Andrew Senior, Mark Z Mao, et al. 2011. Improving the speed of neural networks on CPUs. In *Proc. deep learning and unsupervised feature learning NIPS workshop*, Vol. 1. 4.

[11] Yi Wang, Wenshan Li, Tao Li, and Hao Tian. 2025. Single-stage filter-based local feature selection using an immune algorithm for high-dimensional microarray data. *Applied Soft Computing* 172 (2025), 112895. https://doi.org/10.1016/j.asoc.2025.112895

A HYPERPARAMETER CONFIGURATIONS OF THE IMPROVED MODEL AND ITS COMPARISON MODELS.

A.1 Proposed technique

The hyperparameters for the dynamic implementations can be found in Table 4. ϵ was set to be relatively high to stop the system from taking too long to compress. The stochastic gradient descent optimiser originally had a momentum of 0.4 and norm clipping to 6, and fine-tuned with a learning rate of $8e^{-4}$, momentum of 0.6, and clipnorms of 2.0 and 3.0 for the weaker and stronger configurations, respectively.

A.2 Static SVD

For the weaker static SVD, a pruning threshold ϵ of 0.075 was used. For the stronger one, ϵ was raised to 0.2. For the first epochs, the SGD optimiser had a learning rate of $1e^{-1}$, with a momentum of 0.9 and norm clipping at 2.0. During the post-pruning fine-tuning stage, the optimiser was replaced with one with a lowered learning rate of $2e^{-3}$ and $2e^{-3}$ for the weaker and stronger implementation, respectively, with momentum at 0.4 and norm clipping set to 1.0.

Table 4: Hyperparameter configurations of the improved dynamic systems. μ_{ort} and μ_{norm} were set to be 2 and 1, respectively.

Strength	learning rate	$\mu_{comp,1}$	θ_1	ϵ_1	$\mu_{comp,2}$	θ_2	ϵ_2
Weak	$8e^{-3}$	13	10	0.6	5	5	0.4
Medium	$7e^{-3}$	18	10	0.8	9	5	0.5
Strong	$8e^{-3}$	19	9	0.8	7	5	0.4
None	$6e^{-3}$	0	15	0.5	0	6	0.2

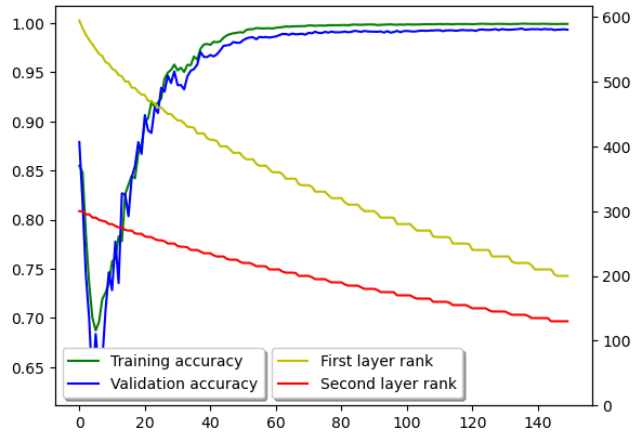

Figure 6: Development of accuracies (left axis) and rank (right axis) in the improved medium dynamic implementation. Similar to other figures, the fine-tuning stage is not included.

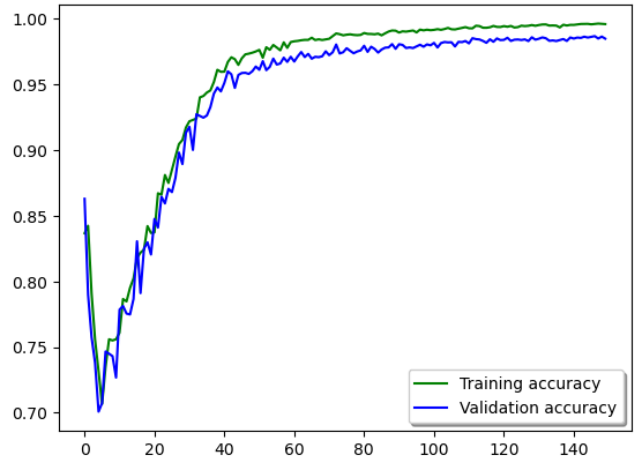

Figure 7: Development of accuracies (left axis) and rank (right axis) in the non-pruning version of the dynamic implementation. Similar to other figures, the fine-tuning stage is not included.

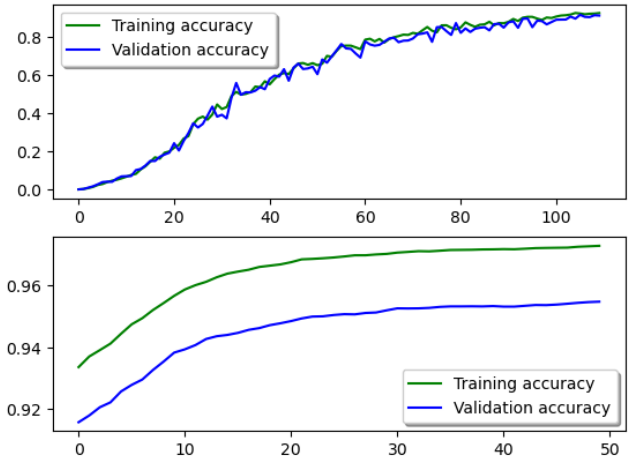

Figure 8: The accuracies of 150-epoch stronger traditional SVD, pre-split (top) and post-split (bottom)

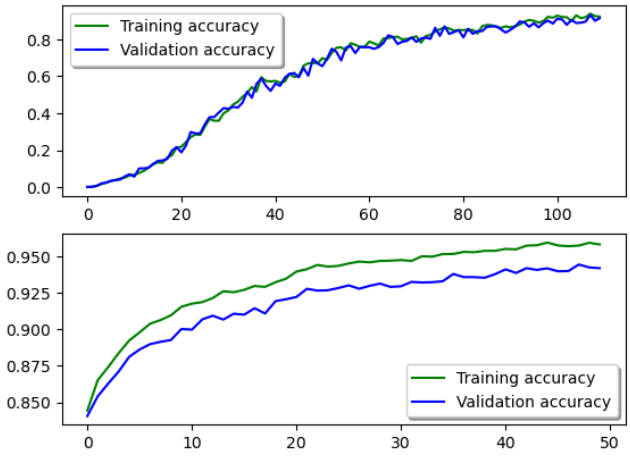

Figure 9: The accuracies of 150-epoch stronger traditional SVD, pre-split (top) and post-split (bottom)

A.3 Control group

In the control group, the initial SGD optimiser had a learning rate of $1e^{-1}$, with a momentum of 0.9 and norm clipping at 3.0 for the larger model and 2.0 for the smaller models. After fine-tuning, the learning rate was set to $3e^{-2}$ and norm clipping to 6.0, while momentum remained at 0.9. In the simpler models, the learning rate started at $1e^{-1}$ while for the fine-tuning it was lowered to $5e^{-3}$, with norm clipping at 1.0 and a momentum of 0.9.

Figure 10: The accuracies of the 150-epoch, normal-sized control model, with main training (top) and fine-tuning (bottom)

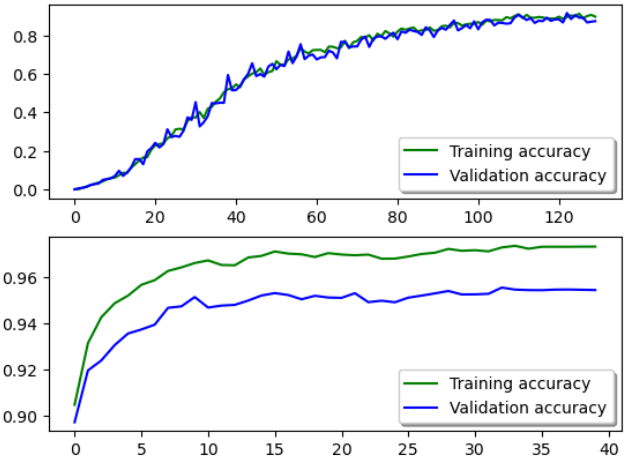

Figure 11: The accuracies of the 150-epoch, smaller control model, with main training (top) and fine-tuning (bottom)

Table 5: Hyperparameter configurations of the original dynamic system. μ_{ort} and μ_{sort} were set to be 2 and 0.5, respectively.

Strength	$\mu_{comp,1}$	θ_1	ϵ_1	$\mu_{comp,2}$	θ_2	ϵ_2
Weak	6	10	0.8	5	5	0.5
Medium	8	12	0.8	6	6	0.7
Strong	14	15	0.8	9	9	0.7

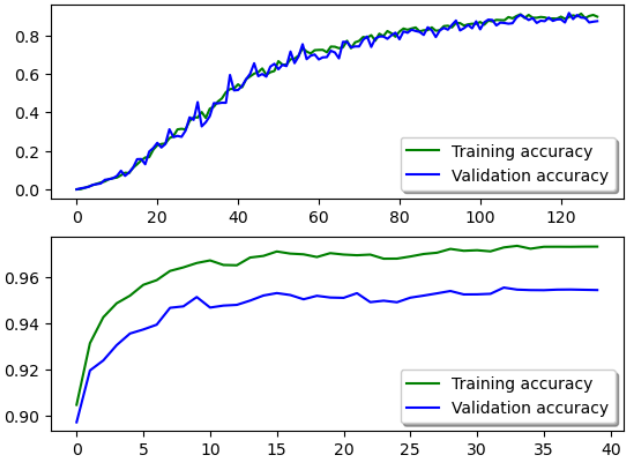

Figure 12: The accuracies of the 150-epoch, smaller control model with layer sizes 100 and 400, with main training (top) and fine-tuning (bottom)

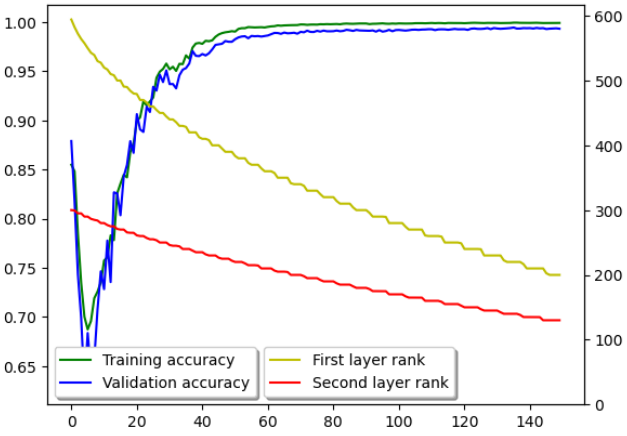

Figure 13: Development of accuracies (left axis) and rank (right axis) in the original medium dynamic implementation. Similar to many other figures, the fine-tuning stage is not included.

B HYPERPARAMETER CONFIGURATIONS OF THE ORIGINAL MODEL AND ITS COMPARISON MODELS.

B.1 Dynamic Technique

The hyperparameters for the dynamic implementations can be found in Table 5. ϵ was set to be relatively high to stop the system from taking too long to compress. The stochastic gradient descent optimiser had a learning rate of $3e^{-3}$, with a momentum of 0.4 and norm clipping to 6.

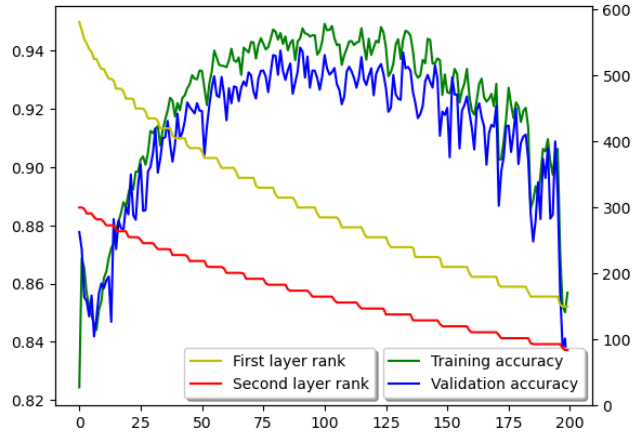

Figure 15: Development of accuracies (left axis) and rank (right axis) in the first strong dynamic implementation. Similar to Figure 2, the fine-tuning stage is not included.

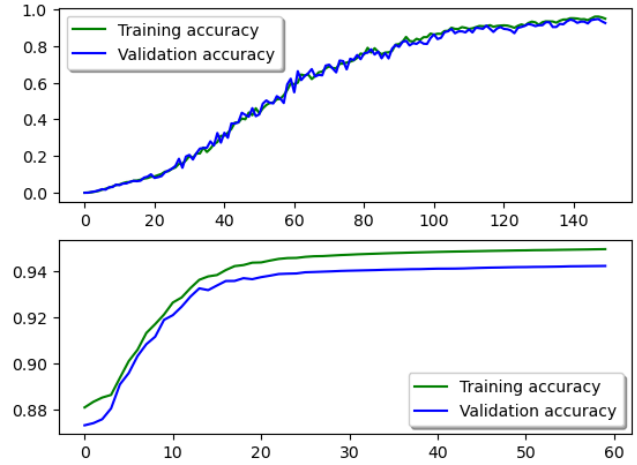

Figure 16: The accuracies of longer trained, weaker traditional SVD, pre-split (top) and post-split (bottom)

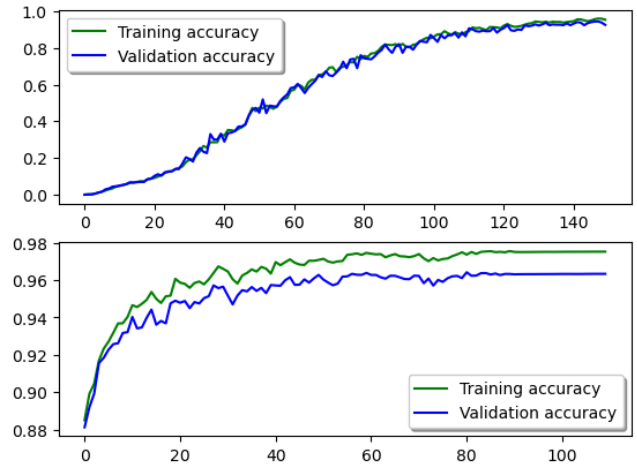

Figure 17: The accuracies of longer-trained, stronger traditional SVD, pre-split (top) and post-split (bottom)

Figure 18: The accuracies of the 200-epoch control model, with main training (top) and fine-tuning (bottom)

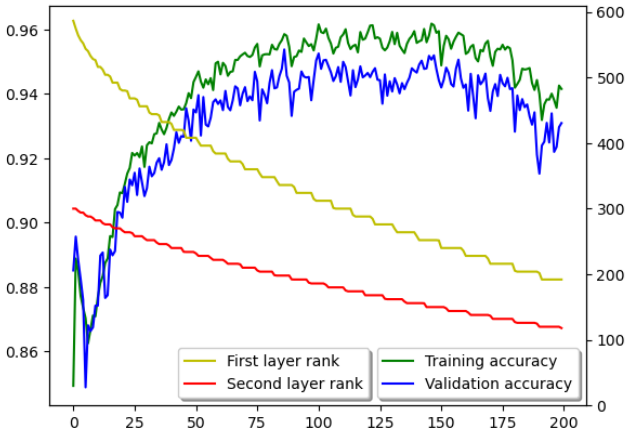

Figure 14: Development of accuracies (left axis) and rank (right axis) in the first medium dynamic implementation. Similar to Figure 2, the fine-tuning stage is not included.

B.2 Static SVD

For the normal static SVD, a pruning threshold ϵ of 0.1 was used. For the more aggressive one, ϵ was raised to 0.3. For the first epochs, SGD optimiser had a learning rate of $1e^{-1}$, also with a momentum of 0.4 and norm clipping to 6. During the post-pruning fine-tuning stage, the optimiser was replaced with one having a lowered learning rate of $5e^{-3}$, increased momentum of 0.9, and norm clipping set to 1.0.

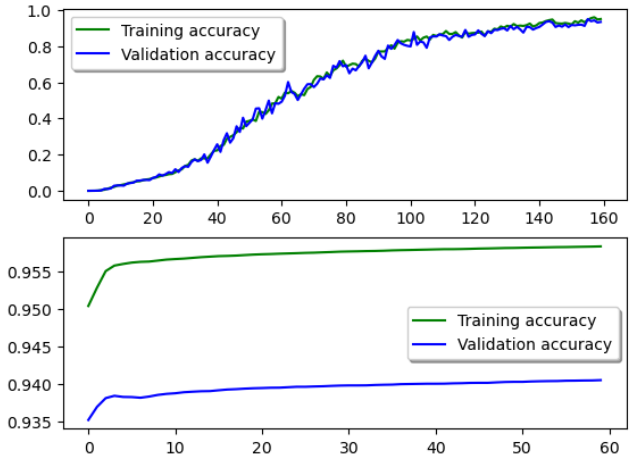

Figure 19: The accuracies of the non-pruning simple model, with main training (top) and fine-tuning (bottom)

B.3 Control group

In the control group, the initial SGD optimiser had a learning rate of $1e^{-1}$, with norm clipping at 2.0 and a momentum of 0.9. After fine-tuning, the learning rate was set to $3e^{-2}$ and norm clipping to 6, while momentum remained at 0.9. In the less complex models, the learning rate started at $1e^{-1}$ while for the fine-tuning it was lowered to $5e^{-3}$, with norm clipping to 1 and a momentum of 0.9