# Enhancing Deep Learning Efficiency: Decomposition Techniques for Neural Network Weights

Authors :

Mehran Tamjidi

Morteza Entezari

Ali Asheri

Ramin Tavakoli

## Project Definition

This project focuses on applying decomposition techniques to the weights of deep neural networks to optimize training efficiency. By breaking down weight matrices into simpler components, the aim is to reduce computational complexity and improve model performance. The study will explore methods like matrix factorization, particularly in MLP networks, to achieve more effective and scalable training. The findings aim to contribute to advancements in deep learning architectures and resource-efficient AI systems.

---

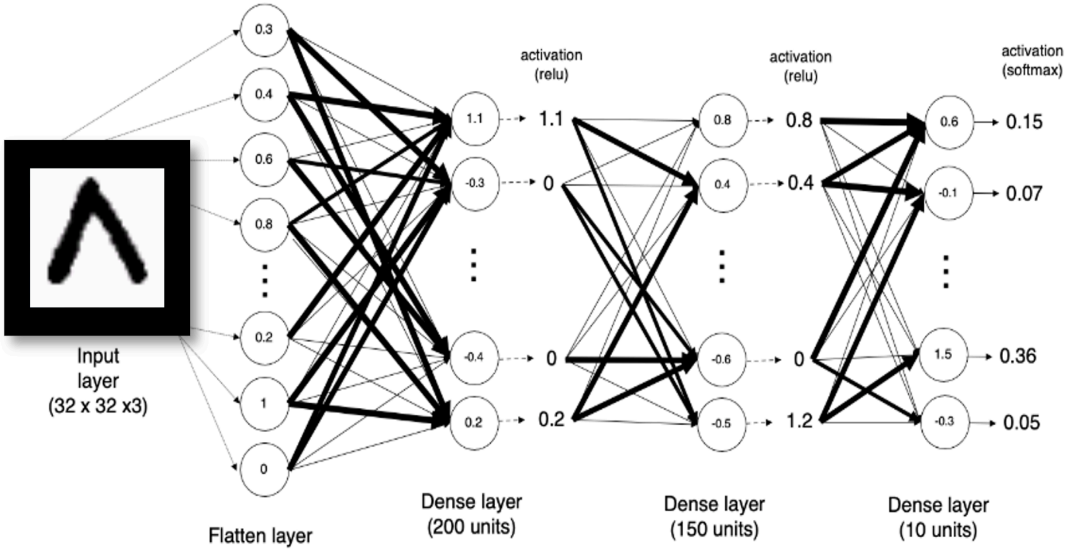
## 1. Datasets

For the first section of this project, you will use the **Hoda Dataset**, a well-known dataset [Link], for Persian handwritten digit recognition. This dataset contains images of Persian digits (۰ to ۹) written by various individuals, making it a valuable resource for training and evaluating machine learning models for digit classification tasks. To simplify the process of loading and preprocessing the dataset, you can use the `DatasetReaders` available on GitHub.

In the second section, you will work with the **Fashion MNIST dataset**, which is a collection of 70,000 grayscale images of fashion items, divided into 10 categories such as T-shirts, trousers, and sneakers. It serves as a drop-in replacement for the classic MNIST dataset, offering more complexity and real-world relevance. The dataset is split into 60,000 training images and 10,000 test images.

For the third section, you will get familiar with the **IMDB dataset**, which is a widely used benchmark for sentiment analysis, containing 50,000 movie reviews labeled as positive or negative. It is evenly split between 25,000 positive and 25,000 negative reviews, making it a balanced binary classification task. The dataset is commonly used to evaluate NLP models' ability to understand and predict sentiment from text.

# 1. Employing SVD for weight decomposition




**Employing SVD for Weight Decomposition and Implementing an MLP in PyTorch**

You are tasked with implementing a **Multi-Layer Perceptron (MLP)** in PyTorch and employing **Singular Value Decomposition (SVD)** for weight decomposition. The goal is to decompose the weight matrices of the MLP using SVD and analyze its impact on model performance.

## Part A: Implement and Train the Architecture in PyTorch

Train the provided Multi-Layer Perceptron (MLP) Architecture on the HODA dataset to achieve the best possible performance. Ensure the model is properly validated to achieve optimal results.

- Before training the model, preprocess the **HODA dataset** by applying normalization and performing data augmentation techniques to improve generalization and performance, if you think it may help.
- Train the network to achieve the best possible performance.
- Ensure the model is properly validated and tuned to achieve optimal results.

## Part B: Apply Singular Value Decomposition (SVD) to the Weight Matrices and retrain the Network with Truncated Weights

- Implement **Singular Value Decomposition (SVD)** on the weight matrices of the trained model.
- In multilayer perceptrons each layer has a linear part in the form of $\mathbf{y = W x + b}$. What we want to do is to decompose the **m** by **n** matrix W into $W = W_1 W_2$, (resulting in 2 linear layers) where $\mathbf{W_1}$ is a **m** by **k** matrix and $\mathbf{W_2}$ is a **k** by **n** matrix. We do this using SVD (low rank approximation you learned in class), where we decompose $\mathbf{W = U \Sigma V^T}$ and truncate as $\mathbf{U[:,:k]\ \Sigma[:k,:k]\ V[:,:k]^T}$. You may set $\mathbf{W_1 = U[:,:k]}$ and $\mathbf{W_2 = \Sigma[:k,:k]\ V[:,:k]^T}$.
- Decompose the weight matrices into their singular values and corresponding singular vectors.
- Truncate the weight matrices by retaining only the top-*k* singular values (where *k* is the rank of the truncated matrix).
- Report the accuracy of the network with the new weights after truncation as a function of k.
- Retrain the network (for each k) using the truncated weight matrices.
- Report the accuracy, memory usage, computation time and speed as a function of k.
- Identify a rather **optimal value of k** balancing between running time and accuracy.

## Part C: Write down the mathematical explanation of how this method works on the provided architecture.

# 2. Fine-tuning the Provided Network Using a Low-Rank Adaptation Method

**Part A: Prepare the Pretrained Model**

- Load the saved pretrained model (from the previous task, part A) that will be fine-tuned using the low-rank adaptation method.

**Part B: Implement the Low-Rank Adaptation Mechanism from Scratch**

1. **Decompose Weight Updates into Low-Rank Components:**
    - Identify the weight matrices in the pre-trained model that will be adapted.
    - Introduce two trainable low-rank matrices (e.g., matrices $A$ and $B$) to approximate updates to the original weight matrices.
    - Initialize the low-rank matrices with appropriate dimensions and small random values.
2. **Modify the Forward Pass:**
    - Adjust the forward pass of the model to incorporate the low-rank updates.
    - Combine the original frozen weights with the low-rank updates during the forward computation.

**Part C: Fine-tune the Model with Low-Rank Adaptation**

- Train the model using the modified architecture with the low-rank adaptation mechanism, in this section you have to finetune the loaded model on **Fashion MNIST dataset** and make sure the input images matches the
- Monitor the training and validation performance to ensure proper convergence.

**Part D: Evaluate the Fine-tuned Model**

- Test the fine-tuned model on the test set and report accuracy, computation time, and memory usage.
- Compare the performance of the fine-tuned model with the original pretrained model. And analyze the impact of the low-rank adaptation method on model efficiency and performance.

**Part E: Repeat the fine-tuning process with different ranks**

- Identify the optimal rank that balances performance and efficiency.
- Report the results for each configuration.

**Part F: Write down the mathematical explanation of how this method works on the provided architecture.**

## 3. Employing SVD for weight decomposition on transformers

### Employing SVD for Weight Decomposition and Implementing an MLP in PyTorch

Traditional models use Recurrent Neural Networks (RNNs) or transformers (e.g., BERT), but transformers have a large number of parameters, making training expensive. To optimize efficiency, we explore using Singular Value Decomposition (SVD) to reduce parameter count in a transformer model while maintaining performance. In this section, you will use the **IMDB dataset**, as discussed previously, to apply and evaluate this optimization approach.

### Part A: Pre-processing

- Tokenize the text using **BERT Tokenizer** or another subword-based tokenizer.
- Use a trainable embedding layer or just use a pre-defined embedding.
- Define a positional encoding. Sinusoidal positional encoding is a common one.
- Feel free to use additional pre-processing steps.

### Part B: Implement a simple multi-head transformer

- Train a simple multi-head transformer Architecture on the IMDB dataset to achieve the best possible performance. Ensure the model is properly validated to achieve optimal results.
- One layer of transformer is sufficient. However, you may try more layers to achieve a possibly better performance.

### Part C: Perform SVD to reduce parameters

1. Initialize attention weight matrices (weights of query, key, value) randomly.
2. Derive SVD of them.
3. Perform truncated SVD to obtain a **k-rank approximation**.
4. Treat the new SVD components as trainable parameters instead of original weight matrices.

5. Train the network again, this time with the new parameters.
6. Identify a rather **optimal value of k** balancing between running time and accuracy.
7. Compare **the running time and accuracy** against those from part B.

**Notes:**

- Report loss values and accuracy for both train set and test set in each epoch.
- Plot loss and accuracy values for both sets at the end of training phase.
- In order to compare results, you must print running time and total parameters of the model. To reach a better observation, you can use **model.named_parameters()** to obtain parameters with name separately.
- A TODO-based structured template [Link] is provided to you. It might help you to start, try to complete it.

**Part D: Write down the mathematical explanation of how this method works on the provided architecture.**

## Final Report

Finally, you should prepare a complete report including:

- All the derivations, equations, and an exhaustive explanation of each part,
- Classification accuracy of your code on the given datasets,
- Plots of confusion matrices, training and evaluation accuracy, and loss curves for each dataset.,

Write down your report in LaTeX.

## Presentation Guidelines

Creating an impactful presentation involves more than just compiling the content; it's about communicating your message effectively and engaging your audience.

# LaTeX Report:

If you're new to LaTeX or need a refresher, go through the tutorial "Learn LaTeX in 30 minutes". This will help you understand the basic commands and structure of a LaTeX document.

Include Essential Elements:

- Your LaTeX document must contain a title, the date, and your name as the author. Make sure these are prominently displayed at the beginning of the document.
- Structure your document with clear sections and subsections. Each major part of your report

Maintain a Consistent and Professional Style:

- Use a consistent style for headings, subheadings, and text. The Overleaf template should provide a good starting point.
- For mathematical notations, use LaTeX's math mode to ensure that equations and formulas are correctly formatted.

Document Your Code and Results:

- Include code snippets where necessary to illustrate how certain parts of the algorithm were implemented. Use the lstlisting environment or similar to format your code.
- Present your results clearly. Use tables for structured data and include plots or graphs where they add value to your explanations. Ensure that all figures and tables have captions and are referenced in the text.

Submission Format:

- Submit a single PDF file containing your complete report. Ensure all text is legible, and all figures and tables are clear and well-labeled.
- In addition to the PDF, submit the .tex source file and any other necessary source files (e.g., images, additional .tex files). This is important for ensuring that your work can be reviewed or edited in its source format.

Proofread and Revise:

- Before submitting, thoroughly proofread your document to correct any typos, grammatical errors, or formatting inconsistencies.
- Ensure that the flow of your document is logical and that each section naturally leads to the next.