NORWEGIAN UNIVERSITY OF LIFE SCIENCES
(NMBU)

DYNAMICAL LOW-RANK TRAINING

OPTIMIZING MEMORY-EFFICIENT NEURAL NETWORKS WITH THE USE OF DIMENSIONALITY REDUCTION TECHNIQUES

AUTHORS

ENDRE ÅSGARD
ERLING MYSEN

NORWAY, FEBRUARY 2026

This page intentionally left blank.

Contents

This page intentionally left blank.

1

Introduction

In the quest for sustainable future, it is important to address the energy demands of AI, especially in deep learning models like ChatGPT. These models, while powerful, demands a high energy costs due to their size and complexity. This report focuses on enhancing energy efficiency in neural networks, exploring methods like dynamical low-rank training and vanilla low-rank to balance computational power and energy consumption, with the goal of moving towards sustainable AI practices.

1.1 Background

A basic understanding of neural networks is fundamental in understanding energy consumption and the low-rank layers. Typically, a neural network with dense layers has a set structure of layers and neurons, tasked with mapping input X to output y, described by $y = f(X)$, where f is the function learned by the network.

1.1.1 forward propagation in a neural network

The equation for the input layer of the network is,

$$Z_1 = XW_1 + b_1\,,$$
$$A_1 = \sigma(Z_1)\,.$$

Which in more general terms, it can be formulated as,

$$Z_l = A_{l-1}W_l + b_l\,, \tag{1.1}$$
$$A_l = \sigma(Z_l)\,. \tag{1.2}$$

In each layer l of the neural network, the output A_{l-1} from the previous layer is multiplied by the weight matrix W_l and added to the bias b_l. This operation, followed by an activation function, forms the core computational step. For our project, the Rectified Linear Unit (ReLU) is used for intermediate layers, scaling outputs between 0 and positive values, while the final layer employs an identity function $f(x) = x$.

1.1.2 Loss function and weight and bias adjustment

The weights in a neural network, symbolizing the connection between layers, are essential for learning the input-output relationship. They, aswell as the biases, are tuned based on the loss function, which assesses the model's accuracy against actual data. In our case, this tuning is guided by the loss calculated on a batch of MNIST images, reflecting the model's deviation from true values and steering weight adjustments.

1.1.3 Backpropagation

Backpropagation is a key step in neural network training, updating weights based on the loss function's performance on a training batch. It calculates the gradient of the loss function with respect to each weight, denoted as $\frac{\partial L}{\partial W}$, where L is the loss function and W the weight matrix. This gradient guides the weight adjustments to improve model performance through training iterations.

The weight update is straightforward:

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial L}{\partial W}, \tag{1.3}$$

where η is the learning rate.

1.1.4 Dynamical vs. Vanilla Low-Rank in neural networks

In neural networks, a dynamical low-rank layer optimizes computational efficiency and accuracy by dynamically adjusting the ranks of matrices, crucial for neural network functions. It replaces a weight matrix W with two smaller matrices U and V, where

$$W \approx U \times V^T. \tag{1.4}$$

These matrices (U in $\mathbb{R}^{m \times r}$ and V in $\mathbb{R}^{n \times r}$) have a lower rank r, substantially reducing parameters, memory usage, and computational time.
Unlike static 'vanilla' low-rank methods which only update matrix values, the dynamical low-rank method adjusts matrix dimensions in response to the loss function, allowing for adaptive control. This maintains learning efficiency in large-scale data scenarios or environments with limited computational resources.

1.2 Creating a neural network in Python using PyTorch

Implementing a neural network in Python with PyTorch requires a solid understanding of object-oriented programming and neural networks. It is a non-trivial task that involves several components. Below is a high-level outline of the Python classes that might be involved in such an implementation:

```
class NeuralNetwork(nn.Module):
    # Neural network logic
```

```
class DataLoader:
    # Data loading and preprocessing logic

class TrainingManager:
    # Training process management logic

class LayerFactory:
    # Layer creation and management logic

class Dense(nn.Module):
    # Dense layer logic

class LowRank(nn.Module):
    # Low-rank layer logic
```

This outline provides a conceptual framework for building a neural network. The key lies in the specific implementations within each class, which will define the network's architecture, learning process, and capabilities.

1.2.1 LowRank class overview

The `LowRank` class extends `nn.Module` and implements a low-rank layer within a neural network using PyTorch. Here is a conceptual overview:

```
class LowRank(nn.Module):
    def __init__(self, dims, activation_function, rank):
        # Initialization with dimensions, activation function, and rank.
        # Definition of U, S, V matrices and bias parameter.

    def forward(self, input):
        # Forward pass through the layer, applying low-rank transformations
        # and activation function.

    def backward(self, learning_rate, s=False):
        # Backward pass for updating parameters using gradients.
        # Implementation of gradient descent for U, S, V matrices and bias.
```

In this class:

- The `__init__` method initializes the layer with specified dimensions, an activation function, and a rank. It also initializes the parameters U, S, V, and bias.
- The `__forward__` method defines the computation performed at every call, applying low-rank matrix multiplication followed by the activation function. Which is a

combination equation 1.4 and 1.1

- The `__backward__` method handles the backpropagation step, updating the parameters based on the description of the given layer earlier.

With this is in mind, the same overall structure applies to the dense and Vanilla layer as well, although there are changes in the computations and initialization.

2

Results

2.1 introduction

This study aimed to evaluate the efficacy of dynamical and vanilla low-rank layers compared to traditional dense layers in neural networks. The results obtained are an indicative of the potential advantages or disadvantages of implementing dynamical or vanilla low-rank layers.

2.2 Reproducing the results

The results presented in this study can be replicated using the provided .toml files which is ordered by the name of the figures, (a,b,c ...), located in the `examples` folder of our project repository, or by Running the `examples_report` script located in the folder. . The files contains all the necessary parameters and settings used to train the neural network, ensuring reproducibility of the reported performance.
Reproducing the results of this study is straightforward by following these steps:

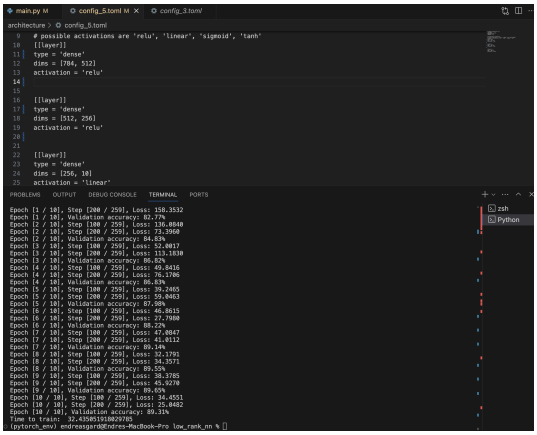
1. Access the `examples` folder in our project repository.
2. Locate and use the files in the folder, or run `examples_report`. This file serves as the configuration input for the training script.
3. Execute the training script using the provided configuration settings.

For a comprehensive guide on running the training script (the .toml files), please refer to the README file in the project repository.

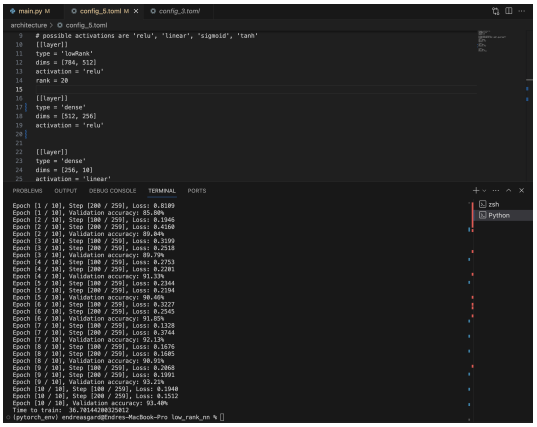
2.2.1 Visualization of model performance

The subsequent images display the outcomes of models trained under uniform settings — a learning rate of 0.001, a batch size of 232, and a duration of 10 epochs. While the neuron count and activation functions remained constant across all models, variations in the rank were introduced. Due to the reduced scale of these images, zooming in may provide a clearer view of the models' performance during both training and validation

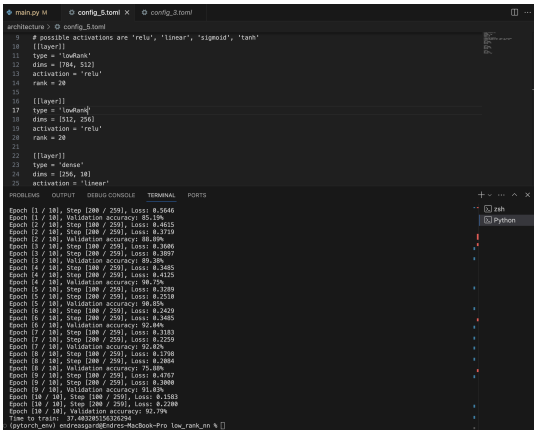
phases. Additionally, each image includes a timer to illustrate the time efficiency of each model in processing the training data.



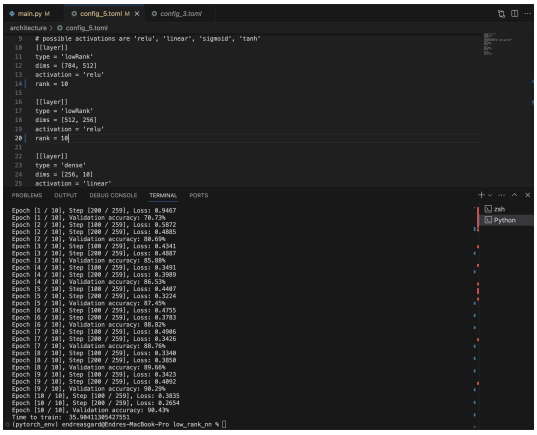
(a) *Model with only dense layers*



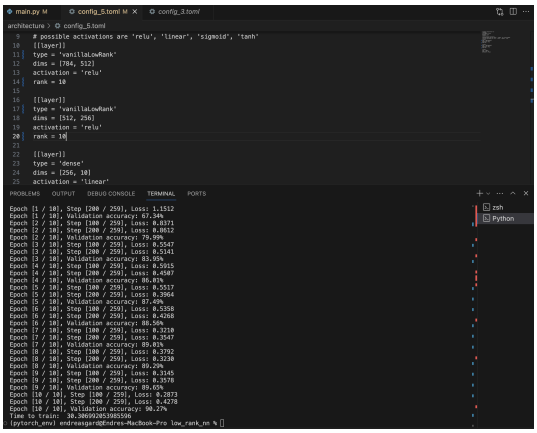
(b) *Lowrank (rank=20) then two dense layers*



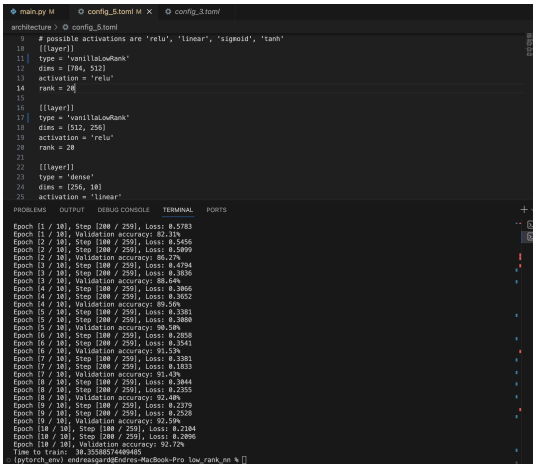
(c) *Two lowrank (rank=20) then dense*



(d) *Two lowrank (rank=10) then dense*



(e) *Two vanillaLowrank (rank=10) then dense*



(f) *Two vanillaLowrank (rank=20) then dense*

Figure 2.1: *Visualization depicting the time complexity of various models and their corresponding performance.*

2.3 Discussion

The data, as illustrated in Figure 2.1a, show that models incorporating low-rank layers demonstrate comparable, if not superior, performance to those with dense layers. Specifically, the models with vanilla low-rank layers exhibited:

- Enhanced computational efficiency, with faster training times compared to dense layer models and dynamical low-rank layers.
- Comparable accuracy in tasks, suggesting that the reduction in complexity does not compromise the model's ability to learn effectively.
- Lower energy consumption, aligning with the goals of sustainable and efficient AI development.

These results underscore the viability of vanilla low-rank layers as an alternative to dense layers, particularly in contexts where computational resource limitations and energy efficiency are paramount. This delicate balance between performance and efficiency brings the potential of low-rank approximations in the design of neural networks.
The figures clearly indicate that models with dynamical low-rank layers, although requiring more training time than those with only dense layers, ultimately demonstrate superior performance. Furthermore, it is evident from the results that the rank plays a crucial role in shaping the performance of both vanilla and dynamical models.

2.4 Sources of error

In assessing these findings, it is important to acknowledge several potential sources of error. A key consideration is the superficial nature of the tests conducted, which is testing on a small dataset.

Variations stemming from the random initialization of parameters are notable. All models were initialized randomly following a normal distribution with values between 0 and 1, which could introduce some level of inconsistency.

Differences in our implementation of layers compared to those in industry-standard libraries may also influence the results.

Additionally, the efficacy of our optimizer must be considered. It may not be as finely tuned for finding the absolute minimum in a high-dimensional graph as other more sophisticated optimizers, potentially impacting the outcomes.

There is a low amount of data the assumptions is being made based off. Which may indicate variation on a bigger scale or study.

3

Conclusion

This study has provided insightful observations into the performance and potential of low-rank layer models in neural networks. The results demonstrate that dynamical and vanilla low-rank layers can serve as effective alternatives to traditional dense layers, particularly in scenarios demanding resource efficiency and energy conservation. Notably, these models have shown promise in terms of computational efficiency and overall performance.

However, it is important to recognize the limitations of our analysis. The study's scope was confined to specific conditions and did not extend to evaluating the performance of low-rank models on large-scale datasets. Therefore, while the findings are promising, they are not conclusive regarding the applicability of these models in broader, more complex real-world scenarios.

Future research should aim to explore the effectiveness of low-rank neural networks in handling larger and more diverse datasets. Such studies could provide a more comprehensive understanding of the practicality and scalability of these models. Additionally, further refinement and comparison with industry-standard implementations could yield deeper insights into the optimal use and integration of low-rank layers in various neural network architectures.

In conclusion, while the results of this study are encouraging, they represent only a early step towards fully understanding and harnessing the capabilities of low-rank neural network models. The journey towards more efficient and powerful AI systems continues, and low-rank approximations remain a promising field for exploration and development.