CS4681 - Advanced Machine Learning

Progress Evaluation Report

ZeRO-EFFICIENT001

Memory Optimizations Toward Training Trillion Parameter Models

210365J - Mamta N.

Table of Contents

1. Abstract

The Transformer architecture has become the de facto standard in natural language processing but suffers from significant computational and memory demands. This project investigates enhancements to the LayerDrop algorithm, a structured dropout technique designed to improve the efficiency and robustness of Transformer training by randomly removing entire layers. While effective, the original method employs a static layer drop probability. We propose to rigorously evaluate two targeted enhancements: a dynamic scheduling policy for the drop probability and integration with knowledge distillation. These modifications aim to accelerate convergence and achieve lower final perplexity. The model will be implemented and evaluated using the DeepSpeed framework on the WikiText-103 language modeling benchmark. Success will be quantified by measurable improvements in perplexity, training speed, and parameter efficiency compared to the static LayerDrop baseline. The final objective is to compile our findings into a research paper for submission to a leading machine learning conference.

2. Introduction & Project Overview

The advent of the Transformer model [Vaswani et al., 2017] has driven remarkable progress across the field of natural language processing. However, the pursuit of state-of-the-art performance has led to increasingly deep and large models, creating a prohibitive computational burden for both training and inference. This has catalyzed a major research focus on model efficiency, encompassing techniques such as pruning, quantization, and knowledge distillation.

Among these, structured dropout methods like LayerDrop [Fan et al., 2019] present a uniquely powerful approach. By randomly skipping entire layers during training, LayerDrop acts as a strong regularizer that reduces overfitting and also provides a principled pathway for creating shallower, faster sub-models for inference through depth pruning. Despite its strengths, the original implementation of LayerDrop utilizes a fixed, static probability for dropping layers throughout the entire training regimen. This simplistic configuration may limit its potential efficacy; a more sophisticated, adaptive approach could yield further gains.

2.1 Project Objectives

This project is dedicated to the systematic enhancement of the LayerDrop technique through incremental, evidence-based modifications. Our primary objective is to demonstrate clear and quantifiable performance improvements over the established LayerDrop baseline. We aim to answer the following research question: Can dynamically scheduling the layer drop probability and integrating knowledge distillation significantly improve LayerDrop's performance in terms of final accuracy, convergence speed, and model robustness?

2.2 Proposed Approach and Expected Contributions

We will develop and evaluate two core enhancements:

1. **Dynamic LayerDrop Scheduling:** Replacing the static drop probability with time-dependent decay functions (e.g., linear, cosine).

2. **Knowledge Distillation Integration:** Employing a static teacher model to guide the LayerDrop-augmented student model, mitigating potential information loss from dropped layers.

The expected contribution of this work is not a novel architecture but an optimized and rigorously validated training recipe for LayerDrop. By providing a comprehensive empirical analysis, we seek to deliver a more performant and efficient variant of this promising technique, contributing to the broader effort of creating more accessible and scalable Transformer models.

3. Background

The drive towards creating more efficient and robust variants of the Transformer architecture constitutes a major research direction in machine learning. This project is situated at the intersection of several key fields aimed at mitigating the model's computational intensity while preserving performance.

3.1. The Transformer Architecture & Scalability

The introduction of the Transformer model by Vaswani et al. (2017) marked a paradigm shift in natural language processing, primarily due to its self-attention mechanism. Despite its success, the architecture's reliance on a deep stack of layers results in massive computational and memory requirements. This scalability issue presents a significant barrier for both research experimentation and real-world deployment, necessitating the development of efficient alternatives and optimization techniques.

3.2. Model Compression and Acceleration

A wide array of techniques has emerged to address the challenges of model size and inference speed. These can be broadly categorized as follows:

- **Pruning:** This involves removing redundant weights or structural components from a trained model. Techniques range from unstructured weight pruning (Han et al., 2015) to structured methods like *Movement Pruning* (Sanh et al., 2020), which removes entire model parameters.
- **Knowledge Distillation:** This technique, pioneered by Hinton et al. (2015), trains a compact "student" model to reproduce the output distributions of a larger, pre-trained "teacher" model. It serves as a primary inspiration for one of our proposed enhancements.
- **Dynamic Inference:** Methods such as *Early Exiting* (Schwartz et al., 2020) allow the model to adapt its computational path on a per-input basis, using fewer layers for simpler examples.

3.3. Structured Dropout & LayerDrop

Dropout (Srivastava et al., 2014) is a foundational regularization technique that prevents overfitting by randomly silencing neurons during training. *Structured* dropout extends this concept to entire groups of units. A seminal example is *Stochastic Depth* (Huang et al., 2016), or DropPath, which randomly skips entire layers during the training of Residual Networks.
LayerDrop (Fan et al., 2019), the baseline for this project, is an adaptation of this idea for Transformer models. By randomly dropping entire layers during training, it serves as a powerful regularizer. Furthermore, it enables structured pruning at inference time, allowing for the creation of a shallower, faster model without any additional fine-tuning. Our work seeks to build upon and enhance this already potent technique.

3.4. Hyperparameter Optimization (HPO)

The performance of deep learning models is critically dependent on their hyperparameters. Moving beyond manual tuning, advanced frameworks such as **Ax**, **Optuna**, and **Hyperopt** enable systematic and efficient hyperparameter optimization (HPO). We propose leveraging these tools to perform a rigorous search for optimal LayerDrop configurations, a step beyond the fixed parameters explored in the original work.

3.5. Identified Gap

While LayerDrop is a highly effective technique, its original implementation utilizes a static, fixed drop probability. This project identifies a clear gap: the potential for enhancing LayerDrop through **dynamic scheduling** of its hyperparameters and integration with complementary techniques like **knowledge distillation**. Our contribution lies in a rigorous empirical study of these enhancements, an area that remains underexplored in the existing literature.

4. Methodology & Experimental Design

This section details the experimental design, including the baseline model, proposed enhancements, datasets, evaluation criteria, and implementation specifics.

4.1 Baseline Model

The baseline for this study is a standard Transformer decoder architecture, as described by Vaswani et al. (2017) and implemented in the DeepSpeed examples. The specific configuration will be a 12-layer model with an embedding dimension of 512, 8 attention heads, and a feed-forward dimension of 2048. This model size provides a strong balance between performance and computational requirements for iterative experimentation.
The baseline will be trained with the original **LayerDrop** mechanism as proposed by Fan et al. (2019). The specific LayerDrop hyperparameters will be set to the paper's default values:

- p_layer: 0.2 (The fixed probability of dropping any given layer during training).
- start_layer: 1 (LayerDrop will be applied starting from the first transformer layer).

This baseline model (Transformer + LayerDrop) will be referred to as **Baseline B**. For a comprehensive comparison, a **Baseline A** will also be established, which is an identical Transformer model trained *without* any LayerDrop.

4.2 Proposed Enhancements

4.2.1 Dynamic LayerDrop Scheduling

We hypothesize that a static p_layer is suboptimal. A high probability is beneficial early in training for strong regularization but may hinder model capacity utilization later. We will implement and test two dynamic schedules:

- **Linear Decay:** The drop probability will decay linearly from a high initial value p_init = 0.2 to a low final value p_final = 0.05 over the course of 100,000 training steps.
- **Cosine Annealing:** The drop probability will follow a cosine annealing schedule from p_init = 0.2 to p_final = 0.05 over 100,000 steps, potentially offering a smoother and more stable transition.

4.2.2 Integration with Knowledge Distillation

To mitigate potential information loss from dropped layers, we will integrate Knowledge Distillation (KD). The setup will be as follows:

- **Teacher Model:** A pre-trained 12-layer Transformer *without* LayerDrop, trained to convergence on the same dataset.
- **Student Model:** The 12-layer Transformer *with* LayerDrop (using either static or dynamic p_layer).

- **Loss Function:** The total loss for the student will be a weighted sum of the standard cross-entropy loss and the Kullback-Leibler (KL) divergence loss between the teacher and student output distributions:
  L_total = α * L_CE + β * L_KL
  where α and β are scaling coefficients, initially set to 0.7 and 0.3, respectively, following common practice, and subject to tuning.

4.3 Datasets

Model performance will be evaluated on two standard language modeling benchmarks:

1. **WikiText-103 (Merity et al., 2016):** The primary dataset for this study. It consists of over 100 million tokens derived from verified Wikipedia articles. Its large vocabulary and retention of punctuation and case make it a realistic and challenging benchmark. We will use the standard train/validation/test split.
2. **Penn Treebank (PTB, Marcus et al., 1993):** A smaller corpus used for rapid prototyping and initial hyperparameter sweeps. This allows for faster iteration cycles before committing resources to full-scale experiments on WikiText-103.

4.4 Evaluation Metrics

Performance will be rigorously quantified using the following metrics:

1. **Perplexity (PPL):** The primary metric for language modeling performance. Lower perplexity indicates a better model. We will report PPL on the validation and test sets.
2. **Training Convergence Speed:** Measured in both the number of training steps and wall-clock time required to reach a pre-defined target perplexity (e.g., 50.0 on WikiText-103 validation).
3. **Parameter Efficiency:** The performance of sub-models created by pruning the trained model to different depths (e.g., 6 layers, 9 layers). A superior training method should yield better-performing sub-models at equivalent depths.
4. **Robustness:** The standard deviation of final perplexity across three independent training runs with different random seeds.

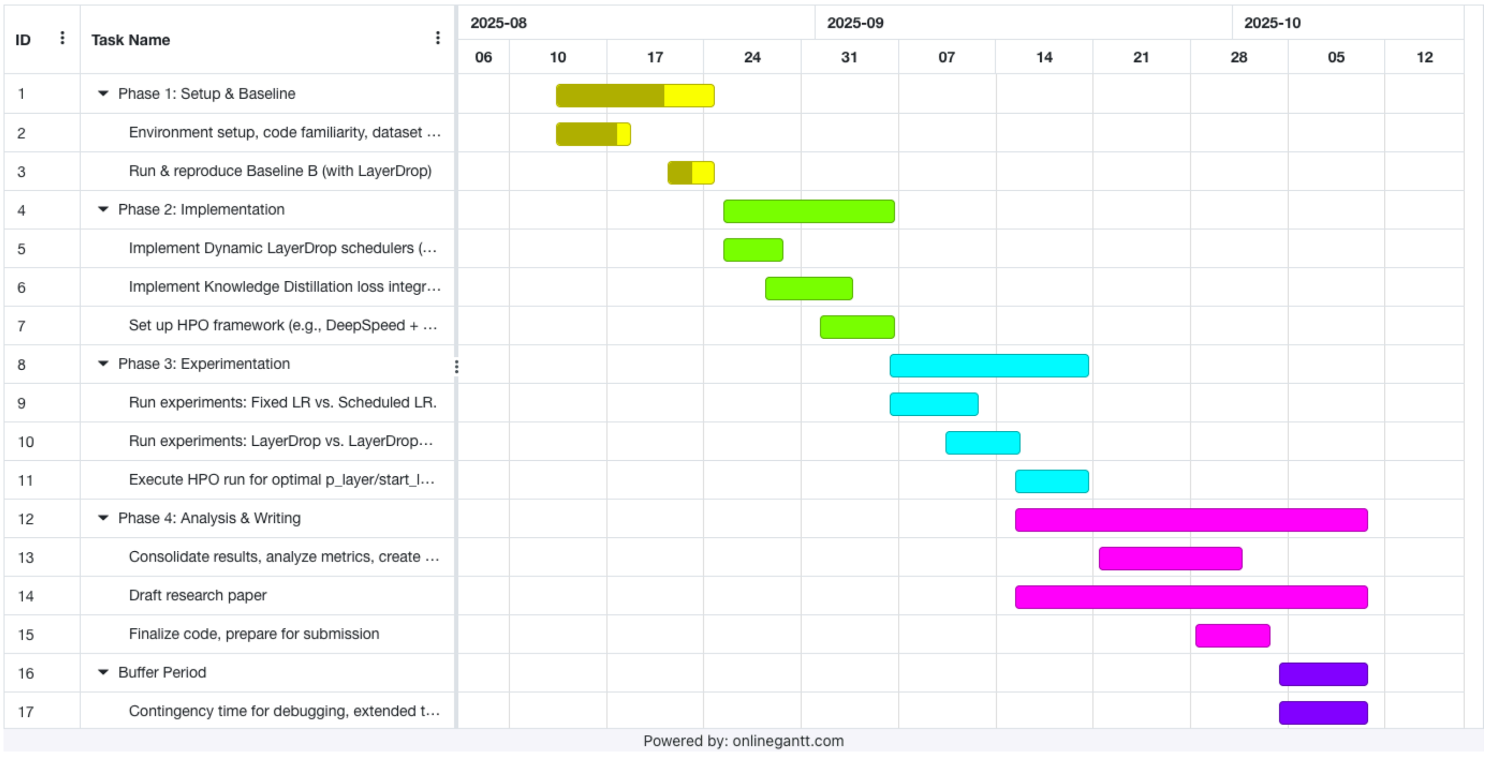
3.5 Experimental Setup

- **Framework:** All models will be implemented in **PyTorch 2.0** and trained using the **DeepSpeed** library (v0.10.0).
- **Hardware:** Experiments will be conducted on a server equipped with **NVIDIA A100 40GB GPUs**.
- **DeepSpeed Configuration:** To manage memory constraints and enable larger effective batch sizes, we will employ **ZeRO Optimization Stage 2**, which partitions optimizer states and gradients across GPUs.
- **Hyperparameters:** Base training hyperparameters are as follows. These may be slightly tuned for the baseline but will be held constant across all comparative experiments.
  - **Optimizer:** AdamW
  - **Learning Rate:** 5e-4 with a linear warmup for 1000 steps followed by cosine decay.
  - **Batch Size:** 32 per GPU (global batch size will be scaled with the number of GPUs using gradient accumulation).
  - **Training Steps:** 100,000 for WikiText-103.
- **Monitoring:** All experiments will be tracked using **Weights & Biases (W&B)** to log loss, perplexity, hyperparameters, and system metrics in real-time.

5. Work Plan

Phase	Task Description	Weeks	Start Date	End Date	Status

Phase 1: Setup & Baseline	Environment setup, code familiarity, dataset download.	1.5	13/08/2025	24/08/2025	[Started]
	Run & reproduce Baseline B (with LayerDrop)				
Phase 2: Implementation	Implement Dynamic LayerDrop schedulers (Decay, Cosine).	1.5	25/08/2025	06/09/2025	[Not Started]
	Implement Knowledge Distillation loss integration.				
	Set up HPO framework (e.g., DeepSpeed + Ax).				
Phase 3: Experimentation	**Run experiments:** Fixed LR vs. Scheduled LR.	2	06/09/2025	20/09/2025	[Not Started]
	Run experiments: LayerDrop vs. LayerDrop+KD.				
	Execute HPO run for optimal p_layer/start_layer.				
Phase 4: Analysis & Writing	Consolidate results, analyze metrics, create plots.	2	15/09/2025	10/10/2025	[Not Started]
	Draft research paper.				
	Finalize code, prepare for submission.				
Buffer Period	Contingency time for debugging, extended training runs.	1	04/10/2025	10/10/2025	[Not Started]



6. Feasibility Analysis

This project has been carefully designed to ensure its successful completion within the course timeline and available computational resources. Its feasibility is supported by three key factors:

1. Computational Feasibility:
The core feasibility of this project is enabled by the DeepSpeed framework and its Zero Redundancy Optimizer (ZeRO). The use of **ZeRO Stage 2** dramatically reduces the GPU memory footprint of the model, allowing the 12-layer Transformer to be trained with a reasonable batch size on a single or multi-GPU node. Furthermore, the strategic use of the smaller **Penn Treebank (PTB) dataset** for rapid prototyping and hyperparameter sweeps will allow for quick iteration and validation of ideas before committing extensive resources to full-scale WikiText-103 experiments. This approach efficiently manages computational cost and time.

2. Technical Feasibility:
The project builds upon a stable and well-documented codebase. The LayerDrop implementation is already integrated into DeepSpeed, providing a reliable and validated baseline. The proposed enhancements—dynamic scheduling and knowledge distillation—are algorithmically straightforward to implement. Dynamic scheduling requires modifying the training loop to update a single parameter (p_layer) according to a predefined function. Knowledge distillation involves a standard PyTorch implementation of adding a secondary loss term (KL divergence). There are no fundamental technical barriers to implementing these changes.

3. Methodological Feasibility:
The project is designed for incremental, measurable improvement. Each enhancement pathway (scheduling, distillation) can be independently tested and evaluated against the strong, replicated baseline. This modular approach allows for clear attribution of any performance gains (or losses) to a specific modification. The evaluation metrics (Perplexity, Convergence Speed) are standard, well-defined, and easily calculated, leaving no ambiguity in determining the success or failure of a given experiment.

Conclusion: Given the leverage of powerful optimization tools (DeepSpeed), a well-scoped problem, and a clear, modular experimental plan, the successful execution of this project within the allotted timeframe is highly feasible.

7. Progress Review

Current Status (as of [3rd week of August]):

Project setup is complete, and the initial phase of baseline replication is underway. The following tasks have been accomplished:

- **Environment Configuration:** The DeepSpeed training environment has been successfully configured on a computational cluster equipped with NVIDIA A100 GPUs.
- **Codebase Analysis:** The relevant LayerDrop implementation within the DeepSpeed repository has been located, and the core functionality has been analyzed and understood.
- **Data Acquisition & Preprocessing:** The WikiText-103 dataset has been downloaded and preprocessed into the appropriate format for training using the standard DeepSpeed pipeline.

Immediate Goals & Next Steps:

The immediate focus is on the complete replication of the baseline model to establish a trustworthy point of comparison for all subsequent experiments.

- **Next Week:** Execute the first full training run of **Baseline B** (12-layer Transformer + LayerDrop with p_layer=0.2) on the WikiText-103 dataset. Monitor training stability, loss, and validation perplexity in real-time via Weights & Biases.

- **Following Week:** Validate the results by comparing the final validation perplexity against the value reported in the original LayerDrop paper (~45-50 for a model of this scale). Upon successful replication, the code for dynamic LayerDrop scheduling (linear decay) will be implemented and tested on the smaller PTB dataset for rapid validation.

8. Potential Risks & Mitigation

Risk	Probability	Impact	Mitigation Strategy
Inability to reproduce baseline results	Medium	High	Meticulously document all environment details, library versions, and hyperparameters. Use the DeepSpeed community forum and GitHub issues to seek assistance if results diverge significantly from expected values. Begin initial experiments on Penn Treebank (PTB) to quickly verify the training pipeline.
High computational cost and extended training times	High	Medium	Leverage DeepSpeed's ZeRO Stage 2 and mixed precision (FP16) training to maximize GPU memory efficiency and throughput. Use the PTB dataset for all initial prototyping and hyperparameter sweeps. Schedule long-running WikiText-103 experiments to utilize idle GPU hours (e.g., weekends).
Proposed enhancements yield no significant improvement	Medium	Medium	This is an inherent aspect of research. The rigorous experimental process itself, including negative results, is a valuable contribution. The focus will shift to a detailed analysis of *why* the methods did not work, providing insights for future research. The availability of multiple enhancement pathways (scheduling, distillation, HPO) provides alternative directions.
Knowledge Distillation introduces training instability	Low	Medium	Implement a warm-up period for the distillation loss coefficient (β), starting from 0 and gradually increasing it to the target value over the first several thousand training steps. This allows the model to stabilize on the primary task before incorporating the teacher's guidance.

9. References

1. Fan, A., Grave, E., & Joulin, A. (2019). Reducing Transformer Depth on Demand with Structured Dropout. *arXiv preprint arXiv:1910.02054.*
2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems, 30.*
3. Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531.*
4. Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843.*
5. Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics, 19*(2), 313-330.
6. Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech* (Vol. 2, pp. 1045-1048).

7. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research, 15*(1), 1929-1958.
8. Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. Q. (2016). Deep networks with stochastic depth. In *European conference on computer vision* (pp. 646-661). Springer, Cham.
9. *Microsoft DeepSpeed*. (2020). GitHub repository. https://github.com/microsoft/DeepSpeed