Optimizing Neural Networks with Language Models

Aarush Gupta

Abstract

Language models (LMs) have demonstrated the ability to automate and enhance numerous tasks, but their potential in meta-optimization remains underexplored. In this paper, we introduce Dux, the first LM-based meta-optimizer designed to accelerate neural network training. By iteratively adjusting optimizer parameters through efficient prompting, Dux outperforms traditional optimizers such as Stochastic Gradient Descent (SGD) and Adam across a diverse set of tasks. We evaluate Dux in both task-agnostic and task-informed conditions. Our results indicate that Dux not only accelerates convergence but also improves generalization, reducing both losses for both training and evaluation datasets. Code at `https://github.com/bxptr/dux`.

1 Introduction

Gradient-based optimization algorithms form the backbone of modern machine learning, enabling the training of deep neural networks across a wide range of applications. Optimizers such as stochastic gradient descent (SGD) and its variants, including momentum-based approaches and adaptive learning rate algorithms like Adam, have significantly advanced the stability, efficiency, and convergence speed of neural network training [2, 13, 9, 3]. However, despite their widespread success, these optimization methods are not universal solutions to all tasks. The performance of any given optimizer often depends heavily on specific characteristics, requiring customization and iterative tuning. The search for optimal hyperparameters, learning rates, and update rules becomes particularly challenging when working in complex decision spaces or with highly non-convex loss landscapes. This is especially true for derivative-free optimization, where gradients are either unavailable or unreliable, making the design and selection of appropriate optimization strategies even more critical.

The necessity of continually refining optimization algorithms highlights a key limitation in the current paradigm: the process is highly task-dependent, often requiring domain-specific knowledge and constant tuning. While advances in algorithmic techniques such as learning rate schedulers and adaptive optimizers have mitigated some of these issues, the need for human intervention remains a bottleneck in achieving efficient and robust optimization. Addressing this challenge requires a more flexible and adaptive approach to optimization, which can dynamically adjust its behavior based on the problem context without requiring constant external intervention.

Recent breakthroughs in language models (LMs) offer a promising avenue for addressing this challenge. large LMs, such as GPT-4o [1] and its successors, have demonstrated remarkable abilities in a variety of natural language tasks via systematic prompting techniques. We consider LMs and not the ubiquitous large language models because

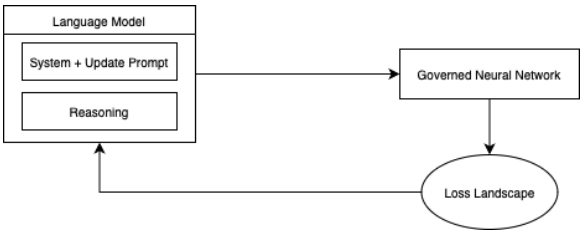

Figure 1: Simplified overview of the Dux optimization process

we hope to generalize this to smaller models. These models have shown proficiency in reasoning, problem-solving, and knowledge synthesis, suggesting that their capabilities could extend beyond conventional language-based tasks. The growing body of research on prompt engineering has underscored the potential of LMs to act as general-purpose meta-optimizers, capable of proposing and refining solutions in computational problems.

In this work, we present Dux (latin: leader, *eg. leading neural networks to convergence*). We explore the hypothesis that large language models can serve as powerful meta-optimizers for neural networks. Specifically, we investigate the use of LMs to iteratively propose novel optimization algorithms and adapt dynamically during the training process. By leveraging meta-cognitive abilities, such as reasoning over optimization steps and providing self-feedback, we aim to create a framework where LMs refine optimization strategies to achieve faster and more robust convergence given training loss metrics and a base optimizer chosen by a human.

2 Problem Setting

We can formalize the meta-optimization problem and its adaptive optimization process as a Markov Decision Process with parameters $(\mathcal{S}, \mathcal{A}, \mathcal{P})$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, and $\mathcal{P}$ represents the state transition probability function. We do not discuss reward since we do not sample the LM in a reinforcement learning setting.

In our meta-optimization framework, the state $s_t \in \mathcal{S}$ at time step t consists of the current state of the model's training, including relevant information any relevant information. The action $a_t \in \mathcal{A}$ at step t corresponds to the modification proposed by the LM to the optimizer configuration. The LM, acting as a meta-optimizer, selects an action a_t based on the observed state s_t to improve model performance.

The state transition is governed by the underlying training dynamics and depends on the optimizer update and dataset of the governed neural network. The state transition function $\mathcal{P}(s_{t+1}|s_t, a_t)$ captures the stochastic nature of training, where the next state s_{t+1} is determined by the current state s_t, the chosen action a_t, and potentially external factors like data variability. This transition includes the model parameter update:

$$\theta_{t+1} = \theta_t - \eta_{t+1} \nabla_\theta \mathcal{L}(\theta_t),$$

where η_{t+1} is the updated learning rate or any other optimizer adjustment proposed by the LLM.

The policy $\pi(a_t|s_t)$ is defined as loosely as a mapping from states to actions. The LM attempts to learn an optimal policy π^* strictly within its latent space and maximizes

the expected cumulative "reward" $\mathcal{R}$, or indications of convergence, over time:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \mathcal{R}(s_t, a_t)\right],$$

The dynamic adaptation of the optimizer ensures that each optimization update is contextually informed by the governed neural networks current state and performance, thereby maximizing the long-term efficacy of the training process. By utilizing the LM to approximate the optimal policy π^*, we can consider the training process as a meta-optimization problem, where the LM continuously adapts the base optimizer to achieve faster and more efficient convergence.

3 Related Works

The field of language models as general meta-optimizers for neural networks has not been explored as deeply and no direct related works exist, as far as authors know. However, language models have been experimented with in the subfield of hyperparameter optimization. Finding optimal hyperparameters is a well-known and notorious problem in machine learning [5, 14, 7, 6]. Initial research largely explored model-free regimes including random and grid search [5], and more advanced methods leveraged multi-fidelity optimization by virtue of optimization being an iterative process [11, 8]. Language models have been introduced as a viable method to hyperparameter optimization [18]; however, previous works do not consider deeper prompt engineering alongside efficient context and entrusting these advanced model's search spaces.

Prompt engineering has become an essential technique in harnessing the full potential of these models. Synonymous with this sub-field includes chain-of-thought prompting [17], zero-shot reasoning [10], and least-to-most prompting [19]. These have expanded the capabilities of LMs, enabling them to tackle more complex and hierarchical tasks. Furthermore, iterative refinement methods, such as self-consistency [16] and self-feedback [12, 4], have demonstrated that LMs can engage in multi-step reasoning, allowing for more sophisticated problem-solving strategies. With modern models, we speculate that many prompting techniques have become natively integrated in proprietary system prompts or within training and develop independent prompting patterns.

4 Methodology

Our proposed method operates within a conventional supervised learning framework, where a machine learning model is trained to minimize a loss function using gradient-based optimization. Instead of using a static optimizer configuration, we introduce the LM as a dynamic meta-optimizer, capable of adapting the optimization process iteratively based on real-time feedback from the training loop. These proposals are continuously integrated, without alteration, into the training loop, resulting in a closed-loop meta-optimization process.

The overall architecture can be described in two main components. Prompt construction, wherein information about the current training state is encoded into a structured prompt that guides the LM in generating optimizer updates, and optimizer proposal generation and integration, wherein the LM outputs a proposed modification to the

Algorithm 1 Dux meta-optimization design

Require: Target neural network $\mathcal{M}$, base optimizer $\mathcal{O}_0$, loss function $\mathcal{L}$, language model `LM`, update prompt `prompt`, initial training parameters η_0, μ_0, and neural network targets $\mathcal{T}$.
for each training step t **do**
    Optimize `LM` with base or previous proposed optimizer $\mathcal{O}_(t-1)$
    Interpolate `prompt` with loss function metrics $\mathcal{L}(\mathcal{M}(x), \mathcal{T}_t)$
    Sample and apply an optimizer proposal with $\mathcal{O}_t \sim$ `LM(prompt)`
end for
Output: optimized neural network $\mathcal{M}$.

optimization process, which is then applied to, or replaces, the base optimizer in real time. The architecture is formalized in Algorithm 1.

Prompting the LM to act as a meta-optimizer requires the construction of two types of prompts: the system prompt and the update prompt. These prompts provide the LM with the necessary information about the training context and recent performance history, enabling it to generate relevant optimizer adjustments.

The system prompt encapsulates the static aspects of the training process, providing the LM with a complete description of the governed neural network, the base optimizer, the loss function, and optionally, the task. This supplies the LM with the broader optimization environment. Specifically, the system prompt includes: (1) a structured description of the neural network architecture, including the number of layers, types of layers (e.g., convolutional, recurrent, etc.), activation functions, and output dimensions (2) a detailed configuration of the base optimizer, which includes the optimizer type (e.g., SGD, Adam), the initial learning rate, and other relevant hyperparameters such as momentum or weight decay, and (3) the details on the loss function, allowing the LM to understand the optimization target and its gradients. In practice, the loss function maintains state, which enables historical proposals to be natively integrated within the traditional instruction-tuned chat format of modern LMs and allows the update prompts to contain performance data that aligns with each proposal.

As training progresses, the LM is prompted with updates that capture the recent performance metrics and trends. This update prompt includes performance data pertaining to the previous proposal, including n random loss values and the average gradient norm. We specifically choose not to employ a look-back window to reduce bias towards optimization of certain inputs. By instructing the LM to ground itself in reasoning through comments within generated code, we forego validation of the optimizer quality and only test for valid executable code.

The LM constantly generates a proposed adjustment to the optimizer configuration. These modifications are primarily in the form of adjustments to the learning rate or introduction or modification of learning rate schedulers, updates to momentum terms or adaptive learning rate mechanisms, or proposals for switching between different optimization algorithms.

MLP on MNIST (Cross Entropy Loss)						
Epoch	Control		Without Task Knowledge		With Task Description	
	Training	Test	Training	Test	Training	Test
1	2.2918	2.2791	2.2899	2.2718	2.2940	2.2802
2	2.2628	2.2416	0.3389	0.1846	0.3398	0.1649
3	2.2149	2.1793	0.5247	0.3286	0.1387	0.1077
4	2.1359	2.0766	0.1253	0.1181	0.0893	0.0977
5	2.0044	1.9059	**0.0738**	**0.0838**	0.0854	0.1071
RNN on Sine (Cross Entropy Loss)						
Epoch	Control		Without Task Knowledge		With Task Description	
	Training	Test	Training	Test	Training	Test
1	0.8409	0.818	0.819	0.8283	0.8112	0.828
2	0.7982	0.791	0.409	0.3992	0.3955	0.4042
3	0.8129	0.7797	0.4524	0.4793	0.2816	0.2981
4	0.7867	0.7776	0.3326	0.3655	0.2555	0.2804
5	0.7582	0.771	0.3498	0.3197	**0.1917**	**0.1826**
Transformer on IMDB Sentiment Analysis (Binary Cross Entropy Loss)						
Epoch	Control		Without Task Knowledge		With Task Knowledge	
	Training	Test	Training	Test	Training	Test
1	0.6114	0.5699	0.6072	0.5640	0.6075	0.5628
2	0.4946	0.5493	0.5912	0.5929	0.4968	0.5447
3	0.4266	0.5431	0.4429	0.4573	0.4583	0.5223
4	0.3647	0.5402	0.3443	0.4675	0.4354	0.5162
5	0.3183	0.5407	**0.2775**	0.5139	0.4198	**0.5152**

Table 1: Training and Test Losses for Experiments

5 Experiments

Here, we present the evaluation results for Dux. Our experiments demonstrate that Dux brings a significant performance gain. To rigorously evaluate our hypothesis, we conducted a series of experiments: the traditional problem of fitting a Multi-Layer Perceptron (MLP) to the MNIST dataset, training a Recurrent Neural Network (RNN) on a sine dataset, and training a Transformer [15] on the IMDB sentiment analysis dataset. The performance of Dux is compared against a control optimizer under two conditions: (1) the LM without task knowledge and (2) LM with a brief task description.

The baseline optimizer is the Stochastic Gradient Descent (SGD) optimizer initialized with a fixed learning rate of 0.01 and momentum of 0.9 for the first two experiments and the Adam optimizer initialized with a learning rate of 0.001 and betas of $\beta_1 = 0.9, \beta_2 = 0.999$. The meta-optimizer is employed under a default task description and another with a brief (one to three words) description of the task. A new optimizer is proposed every epoch. The training and test losses over five epochs for both tasks are presented in Table 1.

In the MLP task, the baseline SGD shows slow convergence with marginal reduction in training loss over five epochs, reflecting suboptimal learning progress. In contrast, Dux without task-specific knowledge accelerates optimization significantly, achieving a lower training loss by the second epoch and demonstrating its effectiveness in tun-

ing hyperparameters without explicit task guidance. Providing a brief task description ("MNIST") yields only marginal improvement, likely due to the simplicity of the task and the limited number of epochs. For the RNN task, similar trends emerge. The control optimizer exhibits slow convergence, while Dux without task knowledge achieves a markedly lower training loss by the fifth epoch. When provided with a brief task description ("RNN over sine dataset"), Dux demonstrates further improvement in optimization efficiency. In the Transformer experiment, the control optimizer again struggles with slow convergence. Both the task-aware ("Small transformer over IMDB sentiment") and task-agnostic versions of Dux perform similarly, with task knowledge providing no significant advantage. We attribute the lack of discernable convergence in the training loss with task knowledge to the non-deterministic nature of LMs and prompt structure.

However, the LM often indicates a strong grasp of training dynamics and predictions. In our experiements in the RNN task, the LM switched constantly between optimizers, hyperparameters, and schedulers. By epoch 2, it identified the Adam optimizer as more viable to convergence and played with hyperparameters before switching to RMS Prop and then AdamW. This constant adaptation is infeasible in current training regimes and meta-optimizers offer dynamic context-based convergence.

Experimental results demonstrate the effectiveness of our methodology. Several key observations emerge from the empirical evaluation. Across both tasks, Dux leads to significantly faster convergence compared to the control optimizer. This is particularly evident in the early epochs, where the we achieve substantial reductions in both training and test losses. We believe the reduction in overfitting evident in lower test losses stems from the LM's ability to balance exploration and exploitation during the optimization process.

6 Conclusion

In this work, we introduced Dux, a novel meta-optimizer based on language models and demonstrated its effectiveness across diverse neural network training tasks. By leveraging LMs to iteratively adjust optimizer parameters, regimes, and scheduling, Dux achieves faster convergence compared to traditional static optimizers. The experimental results consistently show that Dux outperforms baseline optimizers, particularly in the early stages of training.

While Dux shows great promise, especially in optimizing simpler architectures or tasks with limited training epochs, the marginal gains observed in more complex models like Transformers suggest that future work is needed to refine the method for larger-scale problems. Specifically, exploring deeper integration of task knowledge or developing mechanisms for automated task understanding could further enhance Dux's effectiveness in more challenging settings.

Overall, this work represents a significant step toward bridging language-based models with optimization processes, offering a promising interdisciplinary approach that can accelerate and improve neural network training across a wide spectrum of machine learning tasks.

References

[1] OpenAI et al. *GPT-4 Technical Report*. 2024.

[2] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* (1993).

[3] Thomas Bäck and Hans-Paul Schwefel. "An overview of evolutionary algorithms for parameter optimization". In: *Evolutionary computation* (1993).

[4] Yuntao et al. Bai. "Constitutional AI: Harmlessness from AI Feedback". In: *arXiv* (2022).

[5] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *JMLR* (2012).

[6] Bernd et al. Bischl. "Hyperparameter optimization: Foundations, algorithms, best practices". In: *Wiley* (2023).

[7] Matthias Feurer and Frank Hutter. "Hyperparameter optimization". In: *Automated ML* (2019).

[8] Kevin Jamieson and Ameet Talwalkar. "Non-stochastic best arm identification and hyperparameter optimization". In: *AISTATS*. 2016.

[9] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *ICLR*. 2015.

[10] Takeshi et al. Kojima. "Large language models are zero-shot reasoners". In: *arXiv* (2022).

[11] Lisha et al. Li. "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *JMLR* (2017).

[12] Aman et al. Madaan. "Self-Refine: Iterative Refinement with Self-Feedback". In: *arXiv* (2023).

[13] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* (1999).

[14] Jasper et al. Snoek. "Practical bayesian optimization of machine learning algorithms". In: *NeurIPS* (2012).

[15] Ashish et al. Vaswani. "Attention Is All You Need". In: *CoRR* (2017).

[16] Xuezhi et al. Wang. "Self-consistency improves chain of thought reasoning in language models". In: *arXiv* (2022).

[17] Jason et al. Wei. "Chain of thought prompting elicits reasoning in large language models". In: *arXiv* (2022).

[18] Michael R et al. Zhang. *Using Large Language Models for Hyperparameter Optimization*. 2023.

[19] Denny et al. Zhou. "Least-to-most prompting enables complex reasoning in large language models". In: *arXiv* (2022).