Journal of Mathematical Modeling
Vol. -, No. -, 2026, pp. -. Research Article

JMM

Learning the initial step size and backtracking reduction factor for the Armijo line search

Ahmad Kamandi[†]

Department of Mathematics, University of Science and Technology of Mazandaran, Behshahr, Mazandaran, Iran

Email(s): ahmadkamandi@mazust.ac.ir

Abstract. In this paper, we study the sensitivity of the Armijo backtracking line search to its two important parameters, the initial step size and the reduction factor, and propose a GRU-based recurrent model to estimate them adaptively in the steepest descent method. In the structure of the GRU model, we use a low-dimensional feature vector and safeguard the output so as not to lose the convergence properties. The model is trained on a set of strongly convex quadratic problems with varying condition numbers, enabling it to learn patterns that govern effective step-size selection. Numerical experiments show that the learned methods consistently outperform the conventional baselines, namely a grid-search tuned fixed-parameter Armijo rule and two adaptive heuristic strategies, in terms of both iteration counts and function evaluations.

Keywords: Armijo line search, initial step size, backtracking reduction factor, GRU-based recurrent model.
AMS Subject Classification 2020: 46N10, 65K05, 65K10

1 Introduction

For minimizing a continuously differentiable function $f(w) : \mathbb{R}^n \to \mathbb{R}$, a basic algorithm is the steepest descent method that, starting with an initial guess w_0, generates a sequence of approximations $\{w_k\}$ by the following formula:

$$w_{k+1} = w_k - \alpha_k \nabla f(w_k), \tag{1}$$

where $\nabla f(w_k)$ is the gradient of the function at w_k and α_k is the step size. The performance of the steepest descent method depends heavily on the choice of its step length. Most of machine learning algorithms can be viewed as advanced variants of the steepest descent method. Improving the gradient step by rescaling the step or incorporating additional information from previous iterations [2, 5, 9] has

*Corresponding author
Received: 05 June 2026/ Revised: 12 August 2026/ Accepted: 14 August 2026
DOI: 10.22124/jmm.2026.33801.3107

led to methods such as Adam, RMSProp, and Adagrad. The popularity of gradient-based methods in machine learning is not accidental, as noted by many researchers in the literature [2, 7].

When the gradient is Lipschitz continuous with constant L, i.e., for all $u, v \in \mathbb{R}^n$

$$\|\nabla f(u) - \nabla f(v)\| \leq L\|u - v\|, \tag{2}$$

where $L > 0$ is a positive constant, it can be shown that the steepest descent algorithm with a fixed step size converges to a local minimizer of $f(w)$ provided that α_k is chosen in the range $(0, 2/L)$ [10]. This condition is very easy to state, but not so easy to apply. In particular, the constant L is rarely known a priori, and its correct estimation can be quite costly. Thus, the selection of an appropriate fixed step size becomes a challenge.

In the stochastic setting, where the updates are computed using noisy gradient estimates, convergence is usually ensured under the classical Robbins–Monro conditions

$$\sum_k \alpha_k = \infty, \qquad \sum_k \alpha_k^2 < \infty, \tag{3}$$

together with unbiasedness and bounded variance of the stochastic gradient [13]. These requirements force the step sizes to decrease toward zero. Although this guarantees convergence, it also slows the method in later iterations and limits its practical efficiency. Because of this, both deterministic and stochastic steepest-descent methods face difficulties when they rely only on fixed or predefined step-size rules.

Because of these limitations, line-search methods and in particular the Armijo backtracking rule have been widely recommended in the optimization literature. Armijo line search chooses the step size α_k by enforcing a sufficient decrease condition on the objective, without requiring knowledge of the Lipschitz constant or any specific structure in the gradient errors. Under standard smoothness assumptions, it can be shown that steepest descent equipped with the Armijo rule enjoys global convergence, and this result is well established in classical nonlinear optimization texts [11].

However, even when techniques such as the Armijo backtracking scheme are used, the question of determining an appropriate initial step size α_k^0 and step-size reduction factor β_k is still nontrivial. The initial step size has an enormous effect on the behavior of the optimization algorithm. If it is chosen too small, the Armijo condition may be satisfied immediately at every iteration, but the resulting progress per step will be negligible. Conversely, if it is chosen too large, the step size must be reduced substantially to satisfy the condition, incurring a high computational cost.

This trade-off is also observed when picking an appropriate reduction parameter $\beta_k \in (0, 1)$ for backtracking methods. When the β_k is chosen close to 1 the step size will reduce slightly every backtracking step and this may lead to an excessive number of checks at each iteration. On the other hand, a lower β_k decreases the step size faster at each iteration and may cause a small step size to ensure condition satisfaction. Thus both the value of the initial step size and its reduction factor directly affect the performance and efficiency of the algorithm.

Recent advances in meta learning provide a promising direction for addressing the mentioned challenge. In particular, the seminal work of Andrychowicz et al. [1] demonstrated that optimization algorithms can be learned by parameterizing update rules with recurrent neural networks and training them over distributions of optimization tasks. This idea shows that step-size parameters do not always need to be fixed in advance, but can instead be inferred directly from optimization trajectories. However, fully

learned optimizers often lack interpretability and may not retain the robustness and theoretical guarantees of classical line-search methods.

In this work, we adopt a hybrid perspective that combines the strengths of both approaches. Rather than replacing the optimization algorithm entirely, we retain the classical Armijo backtracking framework and focus on learning only its most sensitive and influential components, namely the initial step size α_k^0 and the backtracking reduction factor β_k. This allows us to preserve the stability and theoretical grounding of Armijo line search while introducing data-driven adaptivity through a recurrent model.

The proposed method is based on a GRU architecture that predicts (α_k^0, β_k) from a carefully designed, low dimensional feature representation of the optimization state. In addition, we introduce a trajectory-aware loss function that captures convergence speed, gradient reduction, and backtracking efficiency.

The remainder of the paper is organized as follows. In Section 2 we introduce the recurrent model together with the feature vector and the training loss. In Section 3 the learned steepest descent algorithm is proposed. The numerical results are reported in Section 4, and conclusions are drawn in Section 5.

2 Recurrent model for predicting Armijo parameters

Our goal is to learn how to predict the two parameters, the initial step size α_k^0 and the backtracking reduction factor β_k, of the Armijo line search. So, we are looking for a function Φ that can take the state of the optimization, x_k, and the history of what happened before, h_{k-1}, to predict the desired parameters. i.e.

$$(\alpha_k^0, \beta_k) = \Phi(x_k, h_{k-1}).$$

In machine learning the vector x_k is called the feature vector, constructed from the current optimization state, and h_{k-1} is called the recurrent hidden state that summarizes information from previous iterations.

To realize the mapping Φ, we employ a recurrent neural network based on a GRU module [3]. In our implementation, the GRU is treated as a black-box nonlinear operator provided by the deep learning framework. Its role is to update the hidden state according to

$$h_k = \mathrm{GRU}(x_k, h_{k-1}),$$

without requiring the user to understand or manipulate the internal gating mechanisms. This design keeps the model conceptually simple while still enabling it to capture meaningful temporal dependencies in the optimization trajectory, consistent with prior work demonstrating the effectiveness of recurrent models for learning optimization strategies [1].

Once the hidden state h_k is updated, it is passed through a linear transformation

$$(s_\alpha, s_\beta) = W_{\text{out}} h_k + b_{\text{out}},$$

which produces two raw outputs. These values are then converted into valid Armijo parameters using smooth activation functions:

$$\alpha_k^0 = \mathrm{softplus}(s_\alpha) = \log(1 + e^{s_\alpha}), \qquad \beta_k = 0.01 + 0.89\,\sigma(s_\beta),$$

ensuring that $\alpha_k^0 > 0$ and $\beta_k \in (0.01, 0.9)$. Finally, both quantities are clipped to the ranges

$$\alpha_k^0 \in [10^{-3}, 10], \qquad \beta_k \in [0.01, 0.9],$$

which prevents extreme values and guarantees numerical stability during training and evaluation.

The learnable parameters are the GRU weights and biases together with the output-layer parameters W_{out} and b_{out}. These parameters are important because they decide how the model understands the feature vector x_k. They also decide how the model updates its memory h_k and how it comes up with the predicted Armijo parameters. Despite its simple structure, the model is capable of learning meaningful patterns from optimization trajectories, enabling it to generate adaptive step-size policies that enhance the performance of the Armijo line search.

The choice of a correct feature vector and a proper loss function can be considered crucial to ensure effective training of the GRU network. In particular, the former should contain all the necessary information to predict the values of the Armijo parameters, whereas the latter has to be selected to achieve the desired stability of the training process. This problem is extensively addressed in typical literature sources dealing with sequence modeling [7].

In order to allow for a correct estimation of the Armijo parameters by the recurrent neural network, and to address the issue of scale dependence for better generalization across problems with varying scales, we define a five-dimensional feature vector

$$x_k \in \mathbb{R}^5,$$

that contains the most important features related to the computation of the initial step size α_k^0 and the backtracking reduction parameter β_k. We do not intend to include all gradient/Hessian information into the feature vector, but rather select only those scalar features that play a major role in the performance of the line-search algorithm.

For the sake of simplicity let $f_k = f(w_k)$ and $g_k = \nabla f(w_k)$. To prevent division by zero, a small constant $\varepsilon = 10^{-8}$ is added to the denominators.

The first characteristic captures the relative change in the gradient magnitude compared to the previous iteration:

$$x_k^{(1)} = \frac{\|g_k\|}{\|g_{k-1}\| + \varepsilon}.$$

This ratio allows the model to detect whether the gradient norm is rapidly decreasing, oscillating or increasing.

The second feature captures the relative change in the objective function:

$$x_k^{(2)} = \frac{f_k - f_{k-1}}{|f_{k-1}| + \varepsilon}.$$

This parameter allows the model to understand the relative rate of objective improvement. For instance, if $f_k - f_{k-1}$ is close to zero but the gradient magnitude ratio $x_k^{(1)}$ is still high, the current region is likely ill-conditioned and the steps should be smaller. Otherwise, the steps can be increased provided that f_k decreases monotonically.

Lastly, the remaining three components encode information about the previous choice of the step size:

$$x_k^{(3)} = \alpha_{k-1}, \qquad x_k^{(4)} = \beta_{k-1}, \qquad x_k^{(5)} = \frac{m_{k-1}}{m_{\max}},$$

where α_{k-1} and β_{k-1} denote the accepted step length and the backtracking coefficient of the last iteration, and m_{k-1} stands for the number of backtracking iterations performed to satisfy the Armijo condition,

normalized by the maximum allowed backtracks $m_{\max}$. High values of $x_k^{(5)}$ suggest that the previous initial step size was excessive and needed several corrections, while low values indicate a well-chosen step size.

In sum, this set of five features constitutes a concise and informative input for training our model, capturing the local trend of gradient and objective improvement, as well as the efficiency of previous line-search decisions, without being tied to the absolute scale of the initial state.

The training loss is defined as a multi-component loss of four objectives that help steer the learned line search to perform well, robustly, and efficiently during inference. Let $z_k = \frac{\langle g_k, g_{k-1}\rangle}{\|g_k\|\|g_{k-1}\|}$ denote the cosine similarity between two consecutive gradients. The total loss is given by

$$\mathscr{L} = \lambda_z \left(\frac{1 - z_k}{2}\right) + \lambda_g \left(\frac{\|g_k\|}{\|g_k\| + \|g_0\|}\right) + \lambda_{bt} \frac{m_k}{m_{\max}} + \lambda_{imp} \frac{f_{k+1} - f_k}{|f_k| + |f_{k+1}|}.$$

The first term penalizes the oscillatory behavior of the method. When z_k is close to -1, it shows strong oscillatory behavior, which is often observed in ill-conditioned regions and may suggest the need for more conservative step sizes. In contrast, a value close to $+1$ denotes stable descent directions with little directional change, which may permit a more aggressive step size. The second term helps minimize the gradient norm at each iterate relative to the initial gradient. The third term encourages fewer Armijo backtracking steps by penalizing a large number of reductions. Finally, the fourth term acts as a reward if the function values improve from one iterate to another. Mathematically, for a successful descent step where $f_{k+1} < f_k$, the numerator $(f_{k+1} - f_k)$ is strictly negative, rendering the entire λ_{imp} term negative. Minimizing the total loss $\mathscr{L}$ therefore actively encourages the model to maximize the relative decrease in the objective function at each step. This explicit penalty mechanism prevents the network from converging to a trivial policy that proposes overly conservative step sizes which might easily satisfy the Armijo condition but yield negligible practical improvement in the objective value.

3 Steepest descent with a learned line-search model

In this section, we introduce a learnable version of the steepest descent scheme where the step size rule is defined through a recurrent neural network that learns to produce good step-size parameters from the local properties of the optimization process. In contrast to classic line-search techniques, in which the parameters are fixed in advance, our learned technique makes use of a more flexible step-size rule that depends on the norm of the current gradient, the function values, and the recent line-search feedback.

The trained model is incorporated into the Armijo scheme by the transformation

$$(\alpha_k^0, \beta_k, h_k) = \Phi(x_k, h_{k-1}),$$

resulting in a proposal of the initial step size and the dynamically adjusted reduction factor that depend on the current optimization process status. The Armijo backtrack search loop ensures a sufficient decrease condition and contributes to convergence guarantees under standard assumptions. Feedback data (α_k, β_k, m_k) are then used in the next step by the model to learn from the optimization process dynamics.

Remark 1 (Convergence)**.** *The proposed GRU–Armijo strategy preserves the convergence properties of the classical Armijo backtracking line search. In the proposed algorithm, the GRU model is employed*

Algorithm 1: Steepest Descent with Learned Armijo Parameters

Input: Initial point w_0, trained recurrent model $\Phi(\cdot,\cdot)$, Armijo parameter c.
Output: Approximate solution w^*.
Initialize hidden state $h_0 = 0$.
Initialize memory variables $\alpha_{-1} = \alpha_{\text{init}}$, $\beta_{-1} = \beta_{\text{init}}$, $m_{-1} = 0$.
Set $k = 0$ and initialize $f_{-1} = f(w_0)$, $g_{-1} = \nabla f(w_0)$.
repeat
  Compute $g_k = \nabla f(w_k)$ and $f_k = f(w_k)$.
  Construct the feature vector

$$x_k = \left(\frac{\|g_k\|}{\|g_{k-1}\| + \varepsilon}, \frac{f_k - f_{k-1}}{|f_{k-1}| + \varepsilon}, \alpha_{k-1}, \beta_{k-1}, \frac{m_{k-1}}{m_{\max}} \right).$$

  Predict Armijo parameters using the recurrent model

$$(\alpha_k^0, \beta_k, h_k) = \Phi(x_k, h_{k-1}).$$

  Set $m_k = 0$, $\alpha_k = \alpha_k^0$.
  while $f(w_k - \alpha_k g_k) > f_k - c\alpha_k \|g_k\|^2$ **do**
    $\alpha_k = \beta_k \alpha_k$,
    $m_k = m_k + 1$.
  end while
  Update the iterate $w_{k+1} = w_k - \alpha_k g_k$.
  Increase k by 1.
until a stopping criterion is satisfied.
return $w^* = w_k$.

only to predict the initial trial step size α_k^0 and the reduction factor β_k, whereas the accepted step size is always determined by the standard Armijo backtracking procedure.

Furthermore, the search direction is chosen as

$$p_k = -\nabla f(w_k),$$

which satisfies the sufficient descent and gradient-related conditions. Assuming that the level set

$$S = \{w \in \mathbb{R}^n \mid f(w) \leq f(w_0)\}$$

is bounded and that ∇f is Lipschitz continuous, the assumptions of Theorem 11.7 in [8] are satisfied. Consequently,

$$\lim_{k \to \infty} \|\nabla f(w_k)\| = 0.$$

Therefore, the proposed algorithm inherits the global convergence guarantee of the classical Armijo line search, while the GRU predictor only improves the efficiency of selecting the initial trial step size without affecting the convergence result.

4 Numerical experiments

In this section, we evaluate the performance of the proposed GRU-based learned line search and compare it with several conventional parameter selection strategies. The experiments are conducted on a family of strongly convex quadratic problems with varying condition numbers and dimensions, providing a controlled setting in which the effect of parameter choices can be clearly observed. We first describe the training and evaluation protocols, then report the sensitivity analysis of the loss weights, and finally present comparative results using Dolan–Moré performance profiles.

4.1 Training and evaluation protocols

The model was trained exclusively on 20-dimensional strongly convex quadratic problems generated on the fly. During each of the 150 training epochs, a single new problem

$$f(w) = \frac{1}{2} w^\top A w - b^\top w$$

was constructed, where the symmetric positive definite matrix A had a condition number sampled uniformly from $[5, 200]$. Each optimization trajectory was rolled out for 300 iterations; no convergence-based early stopping was applied during training. The training loss was accumulated over the entire trajectory before a single parameter update was performed. Consequently, each batch consisted of one complete trajectory, yielding 150 optimizer updates and $150 \times 300 = 45{,}000$ optimization-step samples during the whole training phase. Thus, the total number of training trajectories is 150, each containing 300 iterations, and the effective number of training samples is 45,000. No fixed training dataset or separate validation set was used; the training distribution was defined implicitly by the random problem generation process. The network was trained using the Adam optimizer with an initial learning rate of 3×10^{-4} and a cosine annealing scheduler with $T_{\max} = 150$ and $\eta_{\min} = 0$. A global random seed of 123 was set at the beginning of training to ensure reproducibility of the weight initialization and the training dynamics.

For evaluation, the test problems were generated entirely independently and after the training phase had concluded. For each dimension $n \in \{20, 200, 2000\}$, 100 new quadratic problems were generated with condition numbers drawn uniformly from the same interval $[5, 200]$. These test problems were never used during training or for any parameter update. Although the same numerical seed (123) was used to initialize the random number generators in both phases, the training and test problems are distinct because they are produced by different generation procedures: training problems are created sequentially, one per epoch, interleaved with the optimization and gradient computations, whereas the test problems are generated in a separate, dedicated step after training. Therefore, there is no overlap between the problems seen during training and those used for evaluation.

4.2 Implementation details

All algorithms were implemented in Python. The recurrent model was developed using PyTorch [12], while NumPy was used for generating the quadratic test problems. All experiments were performed on a laptop equipped with an Intel(R) Core(TM) i7-7500U CPU @ 2.70 GHz and 12 GB of RAM. The recurrent predictor consists of a single-layer GRU with a hidden-state dimension of 128, followed by a fully connected output layer that predicts the initial trial step size and the backtracking reduction factor.

The network parameters were initialized using Xavier normal initialization for the input-to-hidden weights [6], orthogonal initialization for the hidden-to-hidden recurrent weights, and zero initialization for all bias terms, including those of the output layer.

4.3 Sensitivity analysis

To investigate the sensitivity of the proposed method to the loss weights, we trained the model under nine different configurations of $(\lambda_z, \lambda_g, \lambda_{bt}, \lambda_{imp})$, as listed in Table 1. Based on the numerical results, the *High Improvement* configuration, i.e., $(\lambda_z, \lambda_g, \lambda_{bt}, \lambda_{imp}) = (1,1,1,10)$, yielded the best overall performance in terms of both success rate and function evaluations, and was therefore adopted in all subsequent experiments.

Table 1: Loss weight configurations considered in the sensitivity analysis

Configuration	λ_z	λ_g	λ_{bt}	λ_{imp}
Baseline	1	1	1	1
Only Gradient	0	1	0	0
Only Improvement	0	0	0	1
High Zig-Zag	10	1	1	1
High Gradient	1	10	1	1
High Backtrack	1	1	10	1
High Improvement	1	1	1	10
Grad+Improve	0	1	0	1
Zig+Improve	1	0	0	1

4.4 Baseline methods

To ensure a fair comparison with fixed-parameter Armijo methods, we performed a grid search over the initial step size $\alpha_0 \in \{0.1, 0.5, 1.0, 2.0, 5.0\}$ and the backtracking reduction factor $\beta \in \{0.1, 0.35, 0.5, 0.8\}$ on a separate set of randomly generated problems from the same distribution. The combination that yielded the lowest average number of function evaluations while maintaining the highest success rate was $(\alpha_0, \beta) = (0.5, 0.8)$, which is henceforth referred to as *Tuned*.

In addition, we considered two adaptive heuristic strategies that dynamically adjust the initial step size at each iteration: *AP* (Adaptive Prev-Step), which sets $\alpha_0^{(k)} = \alpha^{(k-1)}$ and $\beta_k = 0.5$, i.e., the accepted step size from the previous iteration, and *AG* (Adaptive Grad-Inv), which sets $\alpha_0^{(k)} = 1/\|\nabla f(w_k)\|$ and $\beta_k = 0.5$, i.e., the inverse of the current gradient norm. Finally, to assess the contribution of the recurrent architecture, we also trained a memory-free multi-layer perceptron (*MLP*) with two hidden layers of width 128, i.e., the same hidden-state dimension as the GRU model, serving as an ablation baseline. More specifically, the MLP receives the same five-dimensional feature vector x_k as the GRU and predicts the same output pair (α_k^0, β_k) using the same output transformation and clipping. Its architecture consists of two fully connected hidden layers with 128 units and tanh activations, followed by a linear output layer of dimension 2. The MLP was trained with the same loss function, optimizer, learning-rate schedule,

number of epochs, and random seed as the GRU model. The only difference is that it does not maintain a recurrent hidden state, so its prediction at iteration k depends only on the current feature vector x_k.

4.5 Stopping conditions and evaluation metrics

All methods used the Armijo constant $c = 10^{-4}$ and started from the same initial point $w_0 = 0$. The stopping conditions were:

$$\|\nabla f(w_k)\| \leq 10^{-4}\|\nabla f(w_0)\|, \qquad \frac{|f(w_k) - f(w_{k-1})|}{|f(w_{k-1})| + 10^{-8}} < 10^{-10},$$

with a maximum of 1000 iterations allowed per run.

4.6 Results

In order to compare the performance on various test instances, we apply the Dolan–Moré performance profiling approach [4]. Figures 1 and 2 present the results for iteration numbers and the number of function evaluations, respectively.

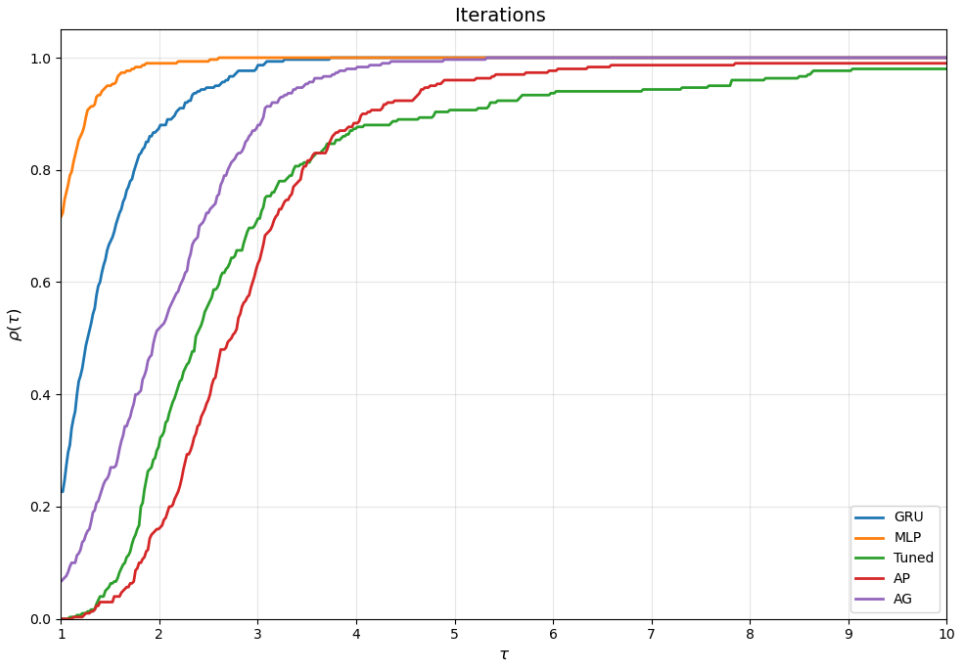

Figure 1: Performance profile for iteration numbers

The Dolan–Moré performance profiles indicate that the learned methods consistently outperform the conventional adaptive baselines. In terms of iterations, MLP achieves the strongest performance, reaching nearly 100% of the test problems within a relatively small performance ratio, while GRU also performs very well and clearly outperforms the Tuned, AP, and AG methods. For function evaluations, GRU provides the best overall performance, followed closely by MLP, with both learned approaches reaching almost complete coverage at substantially smaller performance ratios than the other methods.

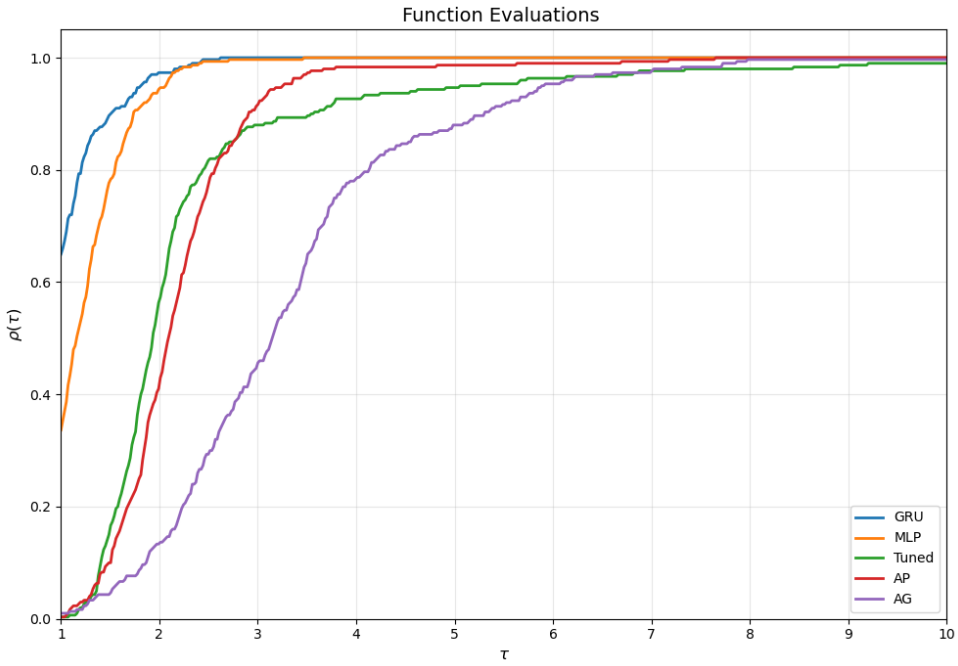

Figure 2: Performance profile for function evaluations

A notable observation from the numerical results is the performance trade-off between the GRU and MLP models. Since the MLP lacks a recurrent hidden state, it relies directly on the current feature vector and may therefore be sufficient for learning an effective initial step size α_k^0, which could explain its strong performance in terms of iteration counts. However, the absence of a learned recurrent representation of the trajectory history might limit its ability to adapt the backtracking policy over successive iterations, potentially leading to more function evaluations.

In contrast, the GRU incorporates a recurrent hidden state that may capture useful temporal patterns in the optimization trajectory. This history-aware representation could help the model adjust α_k^0 and β_k more effectively according to the recent line-search behavior, which may reduce unnecessary backtracking evaluations. Thus, the observed results suggest that recurrent memory may be particularly beneficial for controlling backtracking efficiency, while the explicitly constructed features may already provide sufficient information for effective initial step-size selection.

Overall, these results demonstrate that the learning-based approaches, particularly GRU and MLP, provide superior computational efficiency compared with the conventional methods, with GRU showing a particularly strong advantage when function evaluations are considered.

5 Conclusion

In this paper, we investigated the effect of the two basic parameters of the Armijo line search, the initial step length and the backtracking reduction factor, and showed that their dynamic, state-dependent estimation can substantially improve the efficiency of the steepest descent algorithm. Although the Armijo condition provides a reliable strategy for obtaining a sufficient decrease, the practical efficiency of this

approach depends heavily on the parameter values: a fixed choice of the pair (α_k^0, β_k) tends to result in unnecessary backtracking or overly small step lengths, thus increasing the computational effort required.

To address this problem, we proposed a GRU-based recurrent model that predicts the parameters (α_k^0, β_k) from a short feature vector characterizing the local progress of the optimization process. The predicted parameters are directly integrated into the traditional Armijo procedure, so that its convergence properties are preserved while its efficiency is enhanced in a data-driven manner. The loss weights were justified through dedicated sensitivity analyses, yielding a robust and reproducible training configuration.

Numerical experiments carried out on a class of strongly convex quadratic functions with varying condition numbers and dimensions confirm that the learned methods consistently outperform the conventional baselines, namely the grid-search tuned fixed-parameter Armijo rule (Tuned) and the adaptive heuristic strategies (AP and AG). The Dolan–Moré performance profiles indicate that, in terms of iterations, the MLP ablation achieves the strongest performance, reaching nearly all test problems within a small performance ratio, while the GRU model also clearly outperforms the conventional methods. In terms of function evaluations, the GRU provides the best overall performance, followed closely by the MLP, with both learned approaches reaching almost complete coverage at substantially smaller performance ratios than the other methods.

One limitation of the present study is that the model was trained exclusively on 20-dimensional problems, whereas the numerical tests include dimensions up to 2000. Although the results suggest that the proposed feature representation generalizes reasonably well to larger dimensions, this evaluation involves extrapolation with respect to the problem dimension. Training the model on problems with multiple dimensions and reporting dimension-wise generalization results is a natural direction for future work.

In conclusion, this paper demonstrates that the inclusion of data-driven parameter selection within line-search schemes is a fruitful path for increasing the efficiency of steepest descent methods and their variants. Further research could involve extending this technique to other conditions for line search and to richer feature representations that incorporate curvature information.

References

[1] M. Andrychowicz, M. Denil, S. Gomez, M.W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, N. de Freitas, *Learning to learn by gradient descent by gradient descent*, Adv. Neural Inf. Process. Syst. **29** (2016) 3981–3989.

[2] L. Bottou, F.E. Curtis, J. Nocedal, *Optimization methods for large-scale machine learning*, SIAM Rev. **60** (2018) 223–311.

[3] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, *Learning phrase representations using RNN encoder–decoder for statistical machine translation*, Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP) (2014) 1724–1734.

[4] E.D. Dolan, J.J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program. **91** (2002) 201–213.

[5] J. Duchi, E. Hazan, Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, J. Mach. Learn. Res. **12** (2011) 2121–2159.

[6] X. Glorot, Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, Proc. 13th Int. Conf. Artificial Intelligence and Statistics (AISTATS) **9** (2010) 249–256.

[7] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, 2016.

[8] I. Griva, S.G. Nash, A. Sofer, *Linear and Nonlinear Optimization*, SIAM, Philadelphia, PA, 2009.

[9] D.P. Kingma, J. Ba, *Adam: A method for stochastic optimization*, Proc. 3rd Int. Conf. Learn. Represent. (ICLR) (2015).

[10] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Springer, New York, 2004.

[11] J. Nocedal, S.J. Wright, *Numerical Optimization*, Springer, New York, 2006.

[12] A. Paszke et al., *PyTorch: An imperative style, high-performance deep learning library*, Adv. Neural Inf. Process. Syst. **32**, 2019.

[13] H. Robbins, S. Monro, *A stochastic approximation method*, Ann. Math. Statist. **22** (1951) 400–407.