AdamZ: an enhanced optimisation method for neural network training

ORIGINAL ARTICLE

AdamZ: an enhanced optimisation method for neural network training

Ilia Zaznov[1] · Atta Badii[1] · Julian Kunkel[2] · Alfonso Dufour[3]

Received: 19 March 2025 / Accepted: 3 September 2025

Abstract
AdamZ is an advanced variant of the Adam optimiser, developed to enhance convergence efficiency in neural network training. This optimiser dynamically adjusts the learning rate by incorporating mechanisms to address overshooting and stagnation, which are common challenges in optimisation. Specifically, AdamZ reduces the learning rate when overshooting is detected and increases it during periods of stagnation, utilising hyperparameters such as overshoot and stagnation factors, thresholds, and patience levels to guide these adjustments. While AdamZ may lead to slightly longer training times compared to some other optimisers, it consistently excels in minimising the loss function, making it particularly advantageous for applications where precision is critical. Benchmarking results demonstrate the effectiveness of AdamZ in maintaining optimal learning rates, leading to improved model performance across diverse tasks.

Keywords Machine learning · Deep learning · Neural network training · Multi-head attention layer · Optimisation techniques · Stochastic optimisation · Gradient descent algorithms · Adam optimiser · Dynamic learning rate adjustment · AdamZ optimiser

1 Introduction

In recent years, the machine learning domain has seen significant advancements, particularly in the development of optimisation algorithms that enhance the efficiency and effectiveness of training deep neural networks. Among these algorithms, the Adam optimiser has gained widespread popularity due to its adaptive learning rate capabilities, which enable more efficient convergence compared to traditional methods such as stochastic gradient descent. However, despite its advantages, Adam is not without its limitations, particularly when it comes to handling issues such as overshooting and stagnation during the training process.

To address these challenges, this paper introduces AdamZ as an advanced variant of the Adam optimiser. It is specifically designed to dynamically adjust the learning rate responsive to the characteristics of the loss function, thereby improving both convergence stability and model accuracy. This novel optimiser integrates mechanisms to detect and mitigate overshooting, at the point where the optimiser has stepped too far into the parameter space, and stagnation at the points where progress has started to stall despite ongoing training. By introducing hyperparameters such as overshoot and stagnation factors, thresholds, and patience levels, AdamZ provides a more responsive approach to learning rate adaptation than obtained through Adam.

The development of AdamZ is motivated by the need for more robust optimisation techniques that can adaptively respond to the dynamic nature of neural network training landscapes. Traditional fixed learning rate strategies often struggle with the non-convex and irregular loss surfaces characteristic of deep learning problems,

https://doi.org/10.1007/s00521-025-11649-w

Published online: 16 September 2025

leading to suboptimal performance. By contrast, the AdamZ adaptive strategy enables more precise navigation of these complex landscapes, reducing the likelihood of getting trapped in local minima or experiencing erratic convergence behaviour.

In this paper, we present a comprehensive analysis of the AdamZ optimiser, detailing its theoretical underpinnings and practical implementation. We also provide empirical evidence demonstrating its superior performance in terms of loss minimisation and accuracy of model predictions across a range of benchmark datasets, albeit with slightly longer training times compared to some other optimisers. This trade-off highlights the potential of AdamZ as a valuable tool for applications where accuracy is paramount.

The remainder of this paper is structured as follows: Section 2 reviews the related work in optimisation algorithms for deep learning. Section 3 details the implementation of AdamZ, including its key features and hyperparameters. Section 4 presents experimental results comparing AdamZ to other leading optimisers. Finally, Section 5 discusses the implications of our findings and suggests directions for future research.

2 Related work

The development of optimisation algorithms has been pivotal in advancing the field of machine learning, particularly in training deep neural networks. This section reviews several key optimisers that have influenced the design and functionality of AdamZ.

To provide a structured overview of these methods, a classification tree was constructed based on their key characteristics, as shown in Fig. 1. This tree highlights the relationships between various optimisation methods, offering a high-level summary of their design characteristics.

Each branch represents a decision point that distinguishes methods according to specific features:

1. *Learning rate adaptation*: The root node splits methods into two groups:
   - *Fixed learning rate*: Methods such as SGD [1] and ASGD [2] maintain a constant learning rate throughout training. Fixed learning rates are simple to implement and require fewer hyperparameters to tune. However, they can be inefficient for problems with varying gradient magnitudes, where a constant learning rate may cause slow convergence or overshooting.
   - *Dynamic learning rate*: Methods such a Adagrad [3], RMSprop [4], and Adam [5] adapt the learning rate during training based on gradient information. Dynamic learning rates enable the optimiser to adjust to the scale of the gradients, improving convergence speed and stability. These methods are particularly effective in scenarios where gradients vary significantly across parameters or during different stages of training.

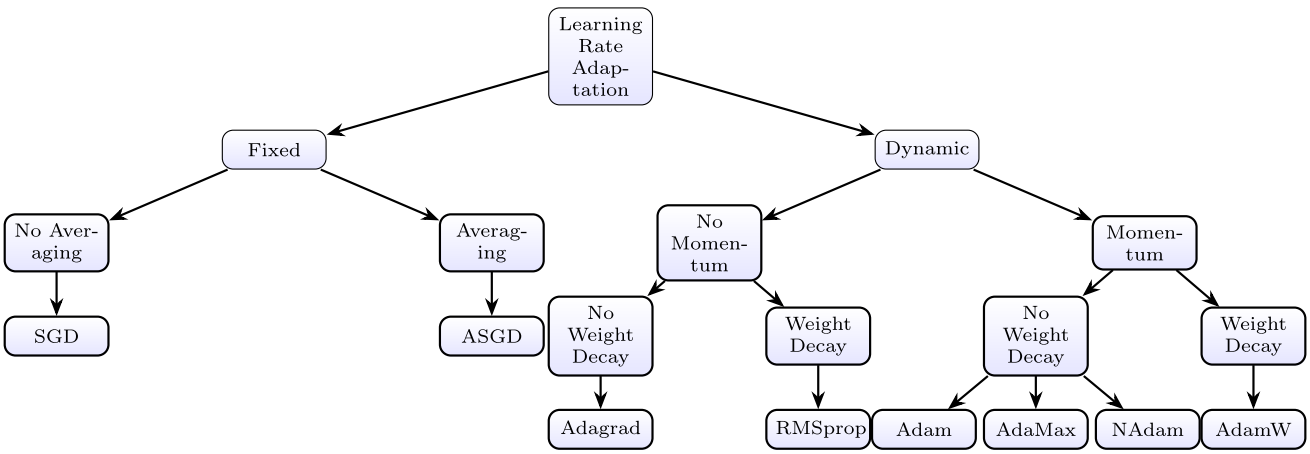

Fig. 1 Classification tree of optimisation methods

2. *Averaging (for fixed learning rate)*: Fixed learning rate methods are further divided based on whether they use averaging techniques:
   - *No averaging*: SGD [1] directly updates weights without averaging. Direct updates are computationally efficient and straightforward but can lead to noisy updates, especially in stochastic settings, requiring careful tuning of the learning rate.
   - *Averaging*: ASGD [2] uses averaging to stabilise updates and improve generalisation. Averaging helps reduce variance in parameter updates, leading to smoother convergence and better generalisation. This is especially useful in non-convex optimisation problems.
3. *Momentum (for dynamic learning rate)*: Dynamic learning rate methods are classified based on whether they incorporate momentum:
   - *No momentum*: Methods such as Adagrad [3] and RMSprop [4] rely solely on adaptive learning rates without momentum. Although these methods are effective in adapting learning rates, they may struggle with optimisation in the presence of noise or poor conditioning, as they lack the acceleration benefits provided by momentum.
   - *Momentum*: Methods such as Adam [5], NAdam [6], and AdaMax [5] include momentum to accelerate convergence. Momentum helps smooth out noisy gradients and accelerates convergence in directions of consistent gradients. It is particularly beneficial for escaping saddle points and navigating ravines in the loss landscape.
4. *Weight decay*: Optimisation methods are further divided based on whether they include weight decay:
   - *No weight decay*: Adam [5], NAdam [6], and AdaMax [5] do not explicitly include weight decay. Without weight decay, these methods may overfit to the training data, especially in over-parametrised models, as they lack explicit regularisation.
   - *Weight decay*: AdamW [7] incorporates weight decay, improving regularisation and reducing overfitting. Weight decay penalises large parameter values, acting as a form of regularisation. This helps prevent overfitting and encourages simpler, more generalisable models.

This classification approach provides a structured framework for understanding the relationships and trade-offs between different optimisation methods. Table 1 summarises the salient characteristics of these optimisers, highlighting their distinctive features and contributions to the field.

Stochastic Gradient Descent (SGD) [1] is a foundational optimisation algorithm introduced in 1951. It updates parameters iteratively by moving in the direction of the negative gradient of the loss function. The update rule is given by:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta)$$

where η is the learning rate. This method is commonly used in training machine learning models, especially in deep learning. It is simple to implement and computationally efficient for large datasets. However, it requires careful tuning of the learning rate and can get stuck in local minima.

Averaged Stochastic Gradient Descent (ASGD) [2], introduced in 1992, enhances convergence by averaging the sequence of iterates. The update rule is:

$$\bar{\theta}_t = \frac{1}{t} \sum_{i=1}^{t} \theta_i$$

Table 1 Summary of key optimisers

Optimiser	References	Year	Description	Hyperparameters	Application
SGD	[1]	1951	Stochastic Gradient Descent (SGD) updates parameters iteratively by moving in the direction of the negative gradient of the loss function: $\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta)$ where η is the learning rate	1 (learning rate)	Requires careful tuning of learning rate
ASGD	[2]	1992	Averaged Stochastic Gradient Descent (ASGD) improves convergence by averaging the sequence of iterates: $\bar{\theta}_t = \frac{1}{t}\sum_{i=1}^{t} \theta_i$	1 (learning rate)	Effective in reducing variance in updates
Adagrad	[3]	2011	Adagrad adapts the learning rate for each parameter based on historical gradients: $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t+\epsilon}} \nabla_\theta J(\theta)$ where G_t is the sum of squares of past gradients	1 (learning rate)	Accumulates squared gradients, which can lead to overly small learning rates
RMSprop	[4]	2012	RMSprop modifies Adagrad by introducing a decay factor: $E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t+\epsilon}} \nabla_\theta J(\theta)$	2 (learning rate, decay rate)	Commonly used in recurrent neural networks
Adam	[5]	2014	Adam combines the advantages of Adagrad and RMSprop, using moving averages of the gradient and its square: $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \; v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \; \hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \; \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon} \hat{m}_t$	3 (learning rate, beta1, beta2)	Widely used due to its robustness and efficiency
AdaMax	[5]	2014	AdaMax is a variant of Adam using the infinity norm: $u_t = \max(\beta_2 u_{t-1}, \lvert g_t \rvert) \theta_{t+1} = \theta_t - \frac{\eta}{u_t} \hat{m}_t$	3 (learning rate, beta1, beta2)	Useful for large parameter spaces
NAdam	[6]	2016	NAdam incorporates Nesterov momentum into Adam: $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon} (\beta_1 \hat{m}_{t-1} + (1-\beta_1) g_t)$	3 (learning rate, beta1, beta2)	Suitable for non-convex optimisation problems
AdamW	[7]	2017	AdamW decouples weight decay from the gradient update: $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon} \hat{m}_t - \eta \lambda \theta_t$	4 (learning rate, beta1, beta2, weight decay)	Suitable for large models with regularisation

This approach is effective in reducing variance in updates and is used in scenarios where reducing the variance of SGD updates is crucial, such as in online learning. However, the averaging process can slow down convergence in the initial stages.

Adagrad (2011) [3] adapted the learning rate for each parameter based on historical gradients. The update rule is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta J(\theta)$$

where G_t is the sum of squares of past gradients and ϵ is a small constant. It is suitable for sparse data and problems with sparse gradients, such as natural language processing tasks. However, it accumulates squared gradients, which can result in a continually decreasing learning rate.

RMSprop (2012) [4] modified Adagrad by introducing a decay factor to control the accumulation of past gradients:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla_\theta J(\theta)$$

RMSprop is popular in training recurrent neural networks and models with non-stationary objectives. It requires tuning of the decay factor and learning rate.

Adam (2014) [5] combined the advantages of Adagrad and RMSprop by using moving averages of the gradient and its square:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
\end{aligned}
$$

This optimiser is widely used across various deep learning applications due to its robust performance. However, it can sometimes lead to non-convergent behaviour or overfitting if not properly tuned.

AdaMax (2014) [5] is a variant of Adam using the infinity norm:

$$
\begin{aligned}
u_t &= \max(\beta_2 u_{t-1}, |g_t|) \\
\theta_{t+1} &= \theta_t - \frac{\eta}{u_t} \hat{m}_t
\end{aligned}
$$

It is particularly useful for models with large parameter spaces.

NAdam (2016) [6] incorporated Nesterov momentum into Adam:

$$
\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} (\beta_1 \hat{m}_{t-1} + (1-\beta_1) g_t)
$$

This optimiser is effective in non-convex optimisation problems.

AdamW [7], introduced in 2017, decouples weight decay from the gradient-based update:

$$
\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t - \eta \lambda \theta_t
$$

It is particularly useful in training large models with regularisation, such as in transformer architectures.

The optimisation methods mentioned are not the only ones available, but they represent foundational and widely adopted approaches. The field of optimisation, particularly as applied to deep learning, is rich and constantly evolving. While methods like SGD, Adam, and their variants are foundational, numerous other approaches and theoretical underpinnings have shaped the landscape. Early contributions established fundamental concepts. Cauchy's work in 1847 [8] laid the groundwork for gradient-based methods, while Polyak's seminal work in 1964 [9] provided crucial insights into momentum and its role in accelerating convergence. Bottou's 1998 work [10] helped establish the framework for stochastic approximation, which is critical for handling the large datasets common in modern machine learning.

Building upon these foundations, researchers have developed a diverse range of optimisation techniques. Nesterov's accelerated gradient method [11] offered an improvement over standard momentum, achieving a faster convergence rate for convex functions. The development of AdaGrad [12] introduced the concept of adaptive learning rates, adjusting them per-parameter based on the historical gradients. RMSProp [13], an unpublished but widely used method often attributed to Geoffrey Hinton, addressed some of AdaGrad's limitations by using a

moving average of squared gradients. More recently, Lion [14] was introduced as a sign-based optimiser, demonstrating competitive performance with Adam.

The quest for efficient and effective optimisation continues, with researchers exploring various avenues. Some focus on adapting learning rates at different granularities, such as AdaLip [15], which performs per-layer adaptation. Others tackle the challenges of distributed training, as seen in AdaGL [16], designed for graph neural networks. Beyond these, significant contributions include the development of methods like the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm and its limited-memory variant L-BFGS [17], which approximate the Hessian matrix to achieve second-order optimisation. The exploration of natural gradients [18] seeks to exploit the Riemannian geometry of the parameter space. AdaBound [19], which dynamically adapts the learning rate bounds to transition smoothly between Adam and SGD, aiming to combine the benefits of both, also falls into this category of adaptive methods.

Further innovations include the introduction of methods specifically designed for non-convex optimisation, such as AMSGrad [20], which addresses convergence issues observed in Adam. Normalised direction-preserving Adam (ND-Adam) [21] proposes modifications to Adam to enhance stability and robustness. Explorations into the relationship between optimisation and generalisation are also crucial, with works like [22] examining the impact of batch size. The development of the Lookahead optimiser [23] introduces a mechanism to improve stability by occasionally interpolating between "fast" and "slow" weights. SGDR (Stochastic Gradient Descent with Restarts) [24] tackles the challenge of escaping local minima by introducing a cyclical learning rate schedule with warm restarts. Finally, works such as Yogi [25] improve the convergence properties and generalisation ability of Adam-style optimisers. K-FAC [26] approximates the Fisher information matrix to achieve more efficient second-order optimisation. Shampoo [27] is a preconditioning method that calculates statistics using higher-order tensors. AdaHessian [28] offers another approach to incorporating second-order information by using a diagonal approximation of the Hessian, aiming to improve convergence speed and generalisation, especially in settings with highly non-convex loss landscapes. NALA (Noise-Adaptive Learning Algorithm) [29], an improved noise-adaptive algorithm that uses mini-batch statistics for faster training, represents a further step in refining adaptive methods for non-convex problems.

2.1 Alternative learning paradigms

While gradient-based optimisers such as SGD, Adam, and RMSprop dominate deep learning, several non-gradient-based optimisation algorithms have been explored in the broader optimisation literature. The most notable among these are Genetic Algorithms (GAs), Particle Swarm Optimisation (PSO), Simulated Annealing (SA), pseudo-inverse modelling, analytical neural networks, and hybrid optimisation approaches.

Genetic Algorithms (GAs) [30] are inspired by the process of natural selection. They operate on a population of candidate solutions, applying selection, crossover, and mutation operators to evolve solutions over generations. GAs are classified as non-gradient optimisers because they do not use gradient information; instead, they rely solely on the evaluation of the objective function. This makes GAs applicable to non-differentiable or highly irregular search spaces. However, GAs are computationally expensive and scale poorly with the number of parameters, making them impractical for deep neural networks with millions of weights.

Particle Swarm Optimisation (PSO) [31] is based on the social behaviour observed in bird flocking or fish schooling. A population of particles explores the search space, with each particle updating its position based on its own best-known position and that of its neighbours. Like GAs, PSO does not require gradient information, relying only on the objective function values. While PSO is robust to noisy or non-differentiable objectives, it suffers from high computational cost and poor scalability in high-dimensional spaces, limiting its applicability in deep learning.

Simulated Annealing (SA) [32] mimics the annealing process in metallurgy. It explores the solution space by probabilistically accepting worse solutions, allowing the algorithm to escape local minima. SA is non-gradient-

based, as it only requires the evaluation of the objective function. However, SA is inherently sequential and converges slowly, especially in high-dimensional settings.

Pseudo-inverse modelling [33], analytical neural networks [34], and hybrid optimisation approaches [35] have shown significant promise in engineering and manufacturing domains. Pseudo-inverse modelling employs analytical solutions to optimise neural networks, circumventing the need for iterative gradient updates. Analytical neural networks leverage closed-form expressions for weight updates, enhancing interpretability and reducing computational overhead. Hybrid optimisation approaches combine gradient-based and non-gradient-based methods, achieving improved stability and convergence in complex systems.

Despite their versatility in non-differentiable or combinatorial problems, non-gradient-based optimisers are not standard in deep learning frameworks for several reasons:

- *Scalability*: Deep neural networks typically have millions of parameters, making population-based or sequential non-gradient methods computationally infeasible.
- *Efficiency*: Gradient-based methods exploit the structure of differentiable loss functions, enabling rapid convergence with far fewer function evaluations.
- *Library support*: Popular libraries such as PyTorch and TensorFlow focus on gradient-based optimisation, and non-gradient methods are not included as standard options.

As a result, comparisons in deep learning research focus on gradient-based optimisers, and non-gradient methods are not considered competitive baselines.

Despite the widespread adoption and success of optimisers such as SGD, Adam, RMSprop, and their variants, each method exhibits notable limitations that can hinder optimal model performance. For instance, SGD, while simple and scalable, often suffers from slow convergence and susceptibility to local minima, especially in highly non-convex loss landscapes. Adam and RMSprop, though adaptive and robust to the choice of learning rates, can exhibit erratic convergence behaviour, overshooting, or stagnation, particularly when hyperparameters are not meticulously tuned. Moreover, these optimisers may struggle with plateaus in the loss surface, leading to inefficient training and suboptimal solutions. These challenges are especially pronounced in deep learning scenarios characterised by high-dimensional, irregular loss landscapes, where the inability to dynamically and appropriately adjust the learning rate can result in either premature convergence or excessive oscillations around minima.

These limitations highlight the necessity for continuous innovation in optimisation methods. The proposed AdamZ optimiser aims to address these issues by enhancing convergence rates and improving the ability to reach global minima. By incorporating novel mechanisms to dynamically adjust learning rates and momentum, AdamZ seeks to provide a more robust and efficient optimisation strategy, paving the way for improved performance in complex, high-dimensional learning tasks. The next section provides a comprehensive overview of the implementation of AdamZ, highlighting its key features and the specific hyperparameters that drive its performance.

3 Implementation of AdamZ optimiser

The AdamZ optimiser is an advanced variant of the traditional Adam optimiser, designed to provide more adaptive learning rate adjustments during the training process. This optimiser addresses two common issues faced in optimisation: overshooting and stagnation. Overshooting occurs when the learning rate is too high, causing the optimiser to miss the optimal point, while stagnation happens when the learning rate is too low, resulting in slow convergence or getting stuck in local minima.

The motivation behind AdamZ is to enhance the flexibility and robustness of the learning process by dynamically adjusting the learning rate based on the behaviour of the loss function. Traditional Adam, while effective, can be deficient in responsively adapting the learning rate given with dynamically changing landscapes of the loss function, leading to inefficient convergence. AdamZ introduces mechanisms to detect and mitigate

these issues by adjusting the learning rate in response to overshooting and stagnation, thereby improving the optimiser's adaptability and efficiency.

Thus, AdamZ incorporates additional hyperparameters that enable it to respond to the training dynamics, as set out below:

- *Overshoot factor* (γ_{over}): Reduces the learning rate once overshooting has been detected, preventing the optimiser from overshooting the minimum.
- *Stagnation factor* (γ_{stag}): Increases the learning rate once loss has started to plateau, thus indicating the onset of stagnation is detected, helping the optimiser to escape local minima.
- *Stagnation threshold* (σ_{stag}): Determines the sensitivity of the stagnation detection based on the standard deviation of the loss.
- *Patience* (p): Number of steps to wait before adjusting the learning rate, allowing for a stable assessment of the loss trend.
- *Stagnation period* (s): Number of steps over which stagnation is assessed.
- *Learning rate bounds* ($\alpha_{\min}$, $\alpha_{\max}$): Ensures that the learning rate remains within this specified range to prevent extreme adjustments.

The above six parameters essentially enable measured i) agility control and ii) over-reactivity control of the AdamZ optimiser in attempting its dynamic responsive adaptation of the learning rate.

The performance of AdamZ is highly dependent on the careful tuning of its hyperparameters. Each hyperparameter plays a critical role in determining how the optimiser reacts to changes in the loss function. Fine-tuning these parameters through techniques such as grid search can significantly enhance the optimiser's performance by adapting it to the specific characteristics of the problem at hand.

The mathematical formulation and algorithmic steps involved in implementing the AdamZ optimiser are set out below:

1. *Initialise parameters:* Initialise the parameters θ_0, and set initial values for the first moment vector $m_0 = 0$, second moment vector $v_0 = 0$, and step $t = 0$.
2. *Hyperparameters:* Define the hyperparameters (note that these are default values inferred from our validation experiments, they must be task/domain-specifically fine-tuned for best performance):
   - Learning rate $\alpha = 0.01$
   - Exponential decay rates for the moment estimates $\beta_1 = 0.9$, $\beta_2 = 0.999$
   - Stability constant $\epsilon = 10^{-8}$
   - Overshoot factor $\gamma_{\text{over}} = 0.5$
   - Stagnation factor $\gamma_{\text{stag}} = 1.2$
   - Stagnation threshold $\sigma_{\text{stag}} = 0.2$
   - Patience period $p = 100$
   - Stagnation period $s = 10$
   - Maximum gradient norm $N_{\max} = 1.0$
   - Minimum and maximum learning rates $\alpha_{\min} = 10^{-7}$, $\alpha_{\max} = 1$
3. *Update rule:* For each iteration t, perform the following steps:
   (a) Compute gradients $g_t = \nabla_\theta f_t(\theta_{t-1})$.
   (b) Update biased first moment estimate:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

   (c) Update biased second raw moment estimate:

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

(d) Compute bias-corrected first moment estimate:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

(e) Compute bias-corrected second raw moment estimate:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

(f) Update parameters:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

4. *Adjust learning rate:* Evaluate the current loss L_t and adjust the learning rate:
   - If overshooting is detected, i.e. $L_t \geq \max(\{L_{t-p}, \ldots, L_t\})$, then:

$$\alpha \leftarrow \alpha \cdot \gamma_{\text{over}}$$

   - If stagnation is detected, i.e. $\text{std}(\{L_{t-s}, \ldots, L_t\}) < \sigma_{\text{stag}} \cdot \text{std}(\{L_{t-p}, \ldots, L_t\})$, then:

$$\alpha \leftarrow \alpha \cdot \gamma_{\text{stag}}$$

   Ensure α is bounded: $\alpha = \max(\alpha_{\min}, \min(\alpha, \alpha_{\max}))$.
5. *Gradient clipping:* Clip the gradients to the maximum norm $N_{\max}$.

Algorithm 1 AdamZ optimiser

Require: Initialise θ_0, $m_0 = 0$, $v_0 = 0$, $t = 0$
Require: Hyperparameters: α, β_1, β_2, ϵ, γ_{over}, γ_{stag}, σ_{stag}, p, s, $\alpha_{\min}$, $\alpha_{\max}$
while not converged **do**
  Compute gradients: $g_t = \nabla_\theta f_t(\theta_{t-1})$
  Update biased first moment: $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
  Update biased second moment: $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  Compute bias-corrected moments:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

  Update parameters:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

  Evaluate loss L_t
  if $L_t \geq \max(L_{t-p}, \ldots, L_t)$ **then**
    $\alpha \leftarrow \alpha \cdot \gamma_{\text{over}}$
  else if $\text{std}(\{L_{t-s}, \ldots, L_t\}) < \sigma_{\text{stag}} \cdot \text{std}(\{L_{t-p}, \ldots, L_t\})$ **then**
    $\alpha \leftarrow \alpha \cdot \gamma_{\text{stag}}$
  end if
  Clip learning rate: $\alpha = \max(\alpha_{\min}, \min(\alpha, \alpha_{\max}))$
end while

To prove the convergence of the AdamZ algorithm, we assume:

- $f(x)$ is Lipschitz smooth: $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$.
- Gradients are bounded: $\|\nabla f(x_t)\| \leq G$.
- Learning rate α_t satisfies:

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty.$$

The zero-bias corrected moments $\hat{m}_t$ and $\hat{v}_t$ are unbiased estimates of the true first and second moments:

$$\mathbb{E}[\hat{m}_t] = \mathbb{E}[\nabla f(x_t)], \quad \mathbb{E}[\hat{v}_t] = \mathbb{E}[(\nabla f(x_t))^2].$$

The update step size is bounded due to the denominator $\sqrt{\hat{v}_t} + \epsilon$:

$$\|x_{t+1} - x_t\| \leq \alpha \frac{\|\hat{m}_t\|}{\sqrt{\hat{v}_t} + \epsilon}.$$

Since $\|\nabla f(x_t)\| \leq G$, we have:

$$\|\hat{m}_t\| \leq G, \quad \sqrt{\hat{v}_t} + \epsilon \geq \epsilon.$$

Thus:

$$\|x_{t+1} - x_t\| \leq \frac{\alpha G}{\epsilon}.$$

The effective learning rate decreases over time:

$$\alpha_t = \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}.$$

As $t \to \infty$, $\hat{v}_t$ stabilises, ensuring that α_t satisfies the conditions for convergence:

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty.$$

Using the smoothness of $f(x)$, we can bound the decrease in the function value:

$$f(x_{t+1}) - f(x_t) \leq -\alpha_t \|\nabla f(x_t)\|^2 + \frac{L}{2}\|x_{t+1} - x_t\|^2.$$

Substituting the bounds for $\|x_{t+1} - x_t\|$ and α_t, we get:

$$\sum_{t=1}^{\infty} \|\nabla f(x_t)\|^2 < \infty.$$

This implies that $\|\nabla f(x_t)\| \to 0$ as $t \to \infty$, i.e. the sequence converges to a stationary point.

AdamZ converges to a stationary point under the assumptions of bounded gradients, Lipschitz smoothness, and appropriate learning rate decay. The key modifications in AdamZ (e.g. overshooting and stagnation adjustment) do not affect the fundamental convergence properties.

The code implementation in Python for the AdamZ optimiser is available at the following GitHub repository: https://github.com/izaznov/AdamZ.git.

Figure 2 illustrates the mechanism of the AdamZ identification of the stagnation and overshooting patterns and respective learning rate adjustments. Initially, periods of stagnation, marked by losses fluctuating around a

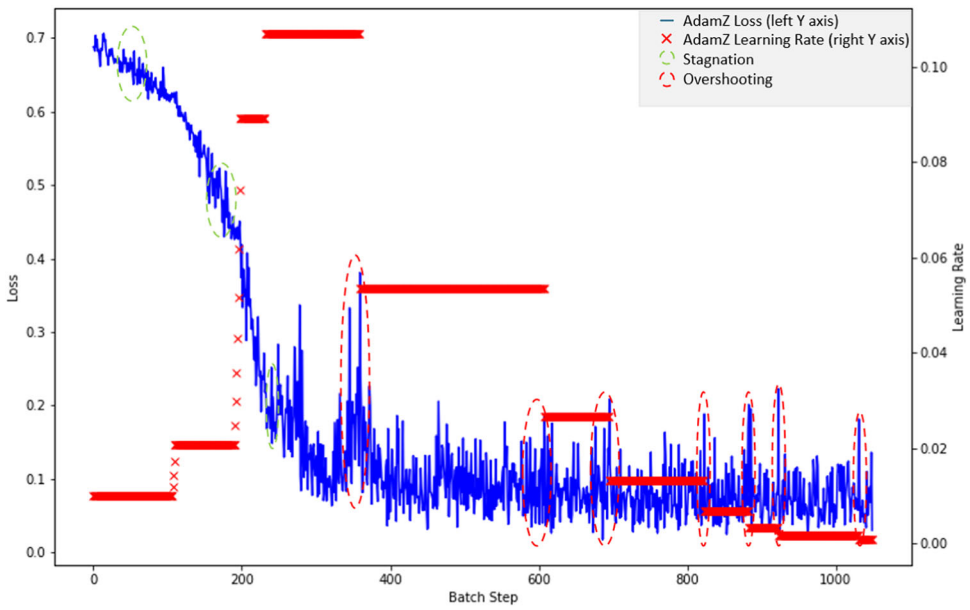

Fig. 2 AdamZ mechanism of detecting and responding to overshooting and stagnation

constant value, trigger three rounds of learning rate increase. Subsequently, spikes in losses indicate overshooting, which in turn trigger seven rounds of learning rate reduction.

The default values of hyperparameters for AdamZ were selected based on a preliminary grid search experiment. The grid search explored combinations of overshoot factors, stagnation factors, stagnation periods, stagnation thresholds, and patience levels to identify configurations that consistently yielded optimal performance. Default values such as $\gamma_{\text{over}} = 0.5$ and $\gamma_{\text{stag}} = 1.2$ were chosen because they demonstrated robust loss minimisation and accuracy improvements.

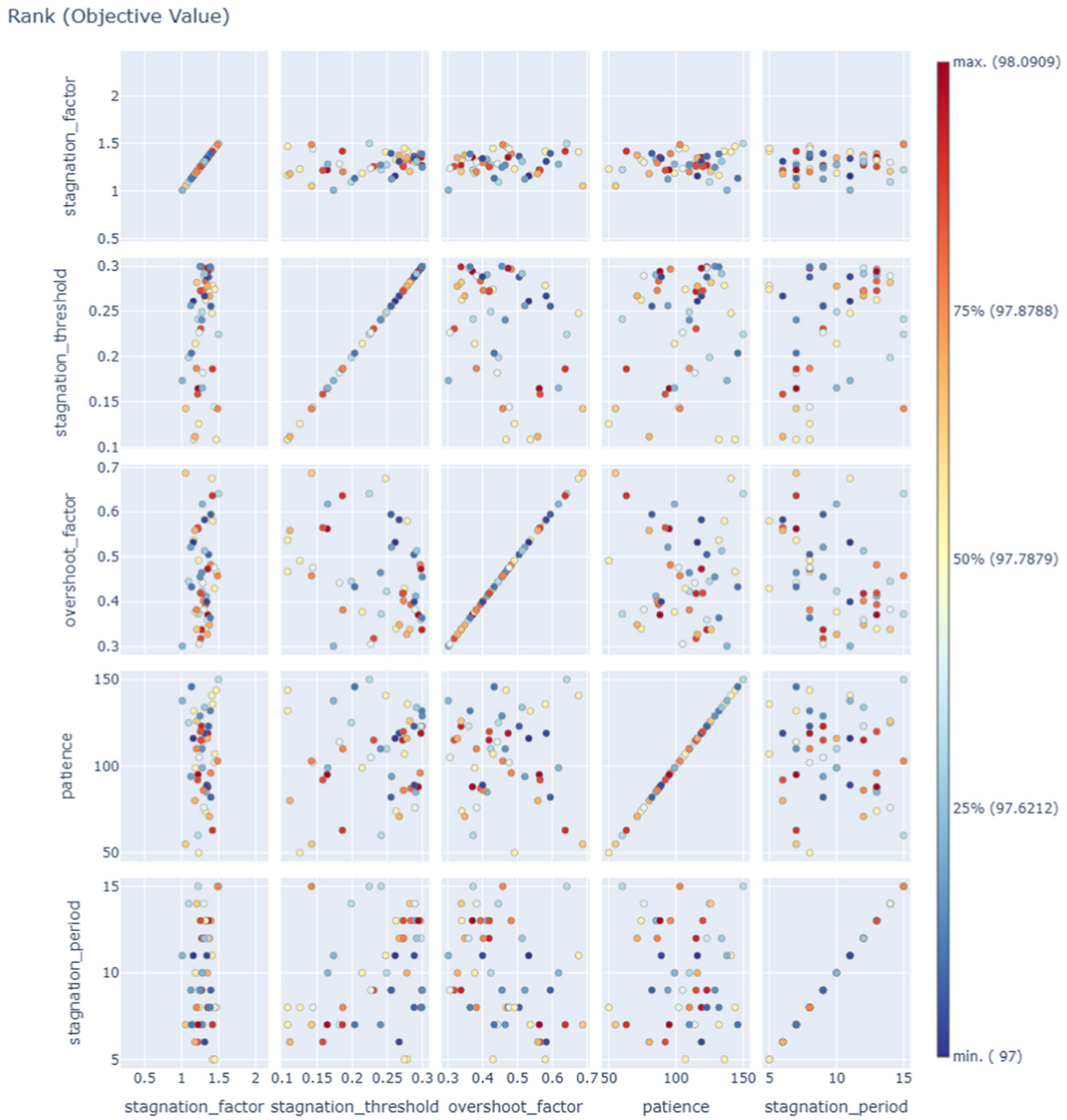

Fig. 3 AdamZ hyperparameters sensitivity and correlation analysis

To further validate the hyperparameter choices, a sensitivity and correlation analysis was conducted to evaluate the impact of varying key hyperparameters on performance. Figure 3 demonstrates that no hyperparameters are strongly correlated, which allows for individual optimisation of each hyperparameter. There are also no substantial swings in the accuracy, which stays in the narrow range of 97-98%, when the hyperparameters fluctuate in a quite broad range.

The introduction of dynamic learning rate adjustments in AdamZ enables it to handle a wider range of optimisation challenges compared to traditional methods. By incorporating mechanisms to detect and respond to overshooting and stagnation, with measured agility to avoid over-reactivity as restrained by the aforementioned control thresholds, AdamZ enhances the convergence speed and stability of the training process. This makes it particularly valuable in complex neural network training scenarios where traditional optimisers may falter.

In conclusion, AdamZ represents a significant advancement in optimisation techniques, offering a more measured and responsive approach to learning rate adjustment. Its effectiveness, however, is contingent upon the appropriate tuning of its hyperparameters, underscoring the importance of experimental validation and parameter optimisation.

4 Experimental results

4.1 Objectives

The primary objective of these experiments was to evaluate the performance of the proposed optimiser, AdamZ, in comparison with other popular optimisation algorithms. We aimed to assess its effectiveness across three different datasets: a synthetic dataset generated using `make_circles` (from sklearn.datasets), the widely used MNIST dataset (from torchvision), and the CIFAR-10 image classification dataset. The implementation of these experiments in Python is available in GitHub repository: https://github.com/izaznov/AdamZ.git, with the first experiment in the file `Circle_adamz_whitepaper.py`, the second in `Mnist_adamz_whitepaper.py`, and the third in `Cifar10_adamz_whitepaper.py`.

These experiments provided insights into the optimiser's performance, including loss, accuracy of prediction, training duration, and overall applicability in neural network training.

4.2 Experiment 1: synthetic dataset using make_circles with a shallow neural network

4.2.1 Dataset description

This experiment utilised the make_circles function from sklearn.datasets to generate a synthetic dataset for binary classification tasks. The controlled nature of this dataset enabled a clear assessment of the optimiser's performance. The make_circles function in the Scikit-learn library was used to generate a synthetic dataset that is particularly useful for binary classification problems. This dataset consisted of two interleaving circles, which made it a challenging test case for algorithms that rely on linear separability. The dataset was generated in a two-dimensional space and was often used to demonstrate the capabilities of non-linear classifiers such as support vector machines with a radial basis function kernel or neural networks.

The make_circles function enabled several parameters to be specified, such as:

- *n_samples*: This parameter defined the total number of samples to be generated.
- *noise*: This parameter specified the standard deviation of Gaussian noise to be added to the data, which can help simulate real-world conditions.
- *factor*: This was the scale factor between the inner and outer circle, determining the relative size of the circles.

Using this dataset, one can effectively test and visualise the performance of classification algorithms that are designed to handle non-linear decision boundaries. Figure 4 illustrates a typical visualisation of the make_circles dataset, showing the inner and outer circles and illustrating the ability of the model to correctly classify whether the particular point belonged to the inner or outer circle.

4.2.2 Model architecture

A shallow neural network architecture was employed, consisting of several fully connected layers. This setup is ideal for quick experimentation and testing on smaller datasets. Figure 5 shows the architecture of the neural network deployed. The hyperparameters were carefully chosen to ensure a fair comparison across different optimisers.

4.2.3 Training setup

The performance of AdamZ was compared against well-established optimisers such as Adam, SGD, and RMSprop, providing a benchmark for evaluating improvements in terms of model classification accuracy, training time, and loss minimisation for 10 epochs (100 steps in each epoch) of training with 100 simulations to account for randomness in the parameter initialisation. The experiment was conducted on a compute cluster with 4x A100 GPUs.

4.2.4 Results

As illustrated in Fig. 6, and summarised in Table 2, AdamZ demonstrated the highest classification accuracy (%), but slightly longer training time (measured in seconds) compared to the other optimisers.

To further validate the robustness of the results, confidence intervals were computed for the accuracy of each optimiser, as shown in Table 3. These intervals provide a statistical measure of reliability, indicating the range within which the true accuracy is likely to fall with 95% confidence.

AdamZ demonstrated the highest median accuracy at 97.83%, with a tight confidence interval ranging from 97.67% to 98%, affirming its consistent performance across simulations. Optimisers like SGD and Adagrad,

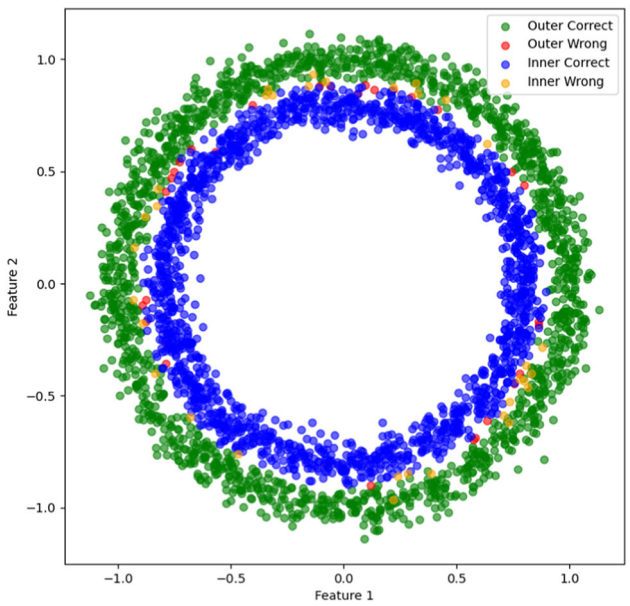

Fig. 4 Visualisation of the make_circles dataset

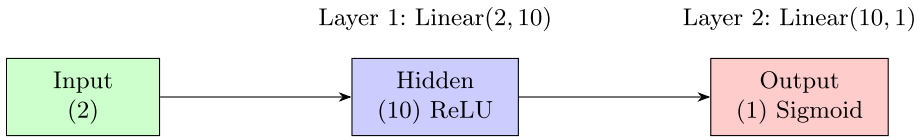

Fig. 5 A neural network architecture applied to the make_circles dataset

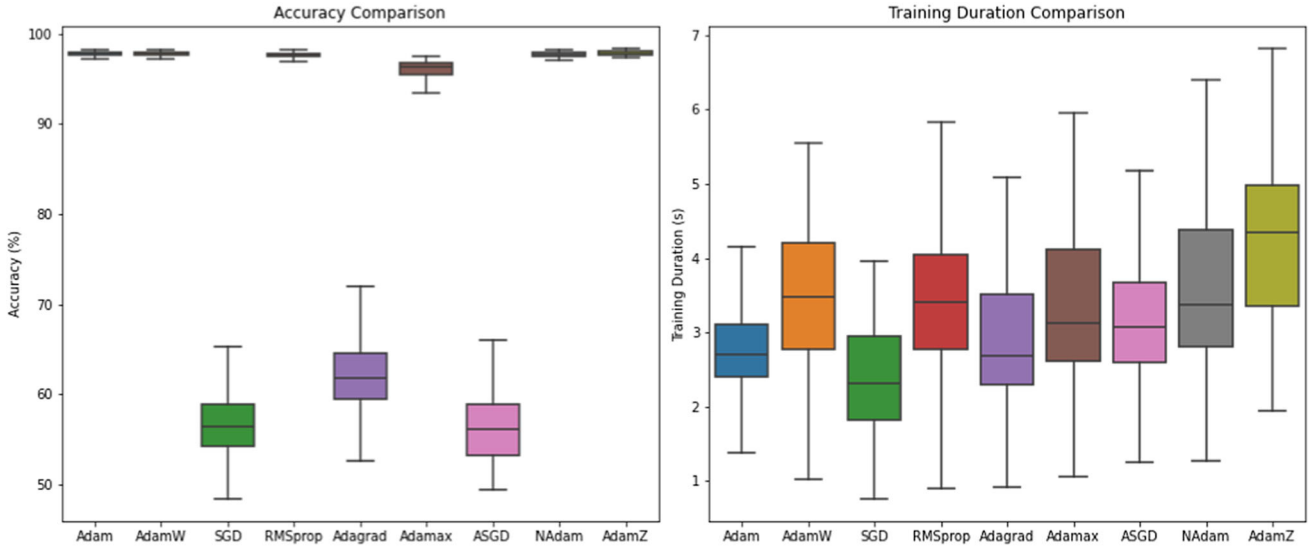

Fig. 6 Accuracy and training duration on the make_circles dataset

Table 2 Performance comparison on make_circles

Optimiser	Accuracy			Training duration		
	Q1	Median	Q4	Q1	Median	Q4
Adam	97.58	97.76	97.85	2.41	2.70	3.11
AdamW	97.58	97.73	97.86	2.78	3.48	4.20
SGD	54.21	56.32	58.84	1.82	2.32	2.94
RMSprop	97.45	97.67	97.82	2.77	3.42	4.04
Adagrad	59.51	61.80	64.50	2.29	2.69	3.52
Adamax	95.39	96.33	96.83	2.62	3.12	4.11
ASGD	53.19	56.14	58.92	2.59	3.07	3.66
NAdam	97.51	97.70	97.88	2.80	3.38	4.38
AdamZ	97.67	97.83	98.00	3.36	4.35	4.99

Table 3 Confidence intervals for accuracy on make_circles

Optimiser	Median accuracy	95% CI Lower	95% CI upper
Adam	97.76	97.62	97.73
AdamW	97.73	97.59	97.73
SGD	56.32	56.05	57.81
RMSprop	97.67	97.49	97.65
Adagrad	61.80	61.55	63.76
Adamax	96.33	95.85	96.27
ASGD	56.14	55.82	57.62
NAdam	97.70	97.58	97.71
AdamZ	97.83	97.75	97.85

which are less suited for non-linear classification problems, exhibited significantly lower median accuracies and wider confidence intervals, highlighting their limitations in handling the `make_circles` dataset.

Considering the training loss evolution of the optimisers in Figure 7, it can be seen that AdamZ is also minimising loss better than other optimisers.

4.3 Experiment 2: MNIST Dataset with a deep neural network

4.3.1 Dataset description

The MNIST dataset is a cornerstone in the field of machine learning and computer vision, commonly utilised for training diverse image processing applications in areas such as document digitization, postal address interpretation, and bank check handling. It was introduced by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges [36]. MNIST stands for Modified National Institute of Standards and Technology database, and it is a large collection of handwritten digits that is commonly used to train image processing systems.

The dataset consists of 70,000 images of handwritten digits from 0 to 9. These images are grayscale and normalised to fit in a 28x28 pixel bounding box, preserving the aspect ratio of the original digit. This normalisation process ensures that the dataset is consistent and easy to use for various machine-learning models.

Key features of the MNIST dataset include:

- *Training set*: 60,000 images used for training models.
- *Test set*: 10,000 images used for evaluating model performance.
- *Balanced classes*: Each digit class is represented equally, providing a balanced dataset for training.
- *Preprocessing*: The digits have been size-normalised and centred in a fixed-size image, making preprocessing minimal.

Figure 8 illustrates the MNIST dataset, showcasing examples of handwritten digits from 0 to 9. Each digit is labelled with its true class, and the predictions made by the model are displayed alongside. This figure demonstrates the digit classification task, whereby a neural network model is trained to accurately recognise and classify each digit, highlighting the role of the dataset in evaluating model performance and optimisation strategies.

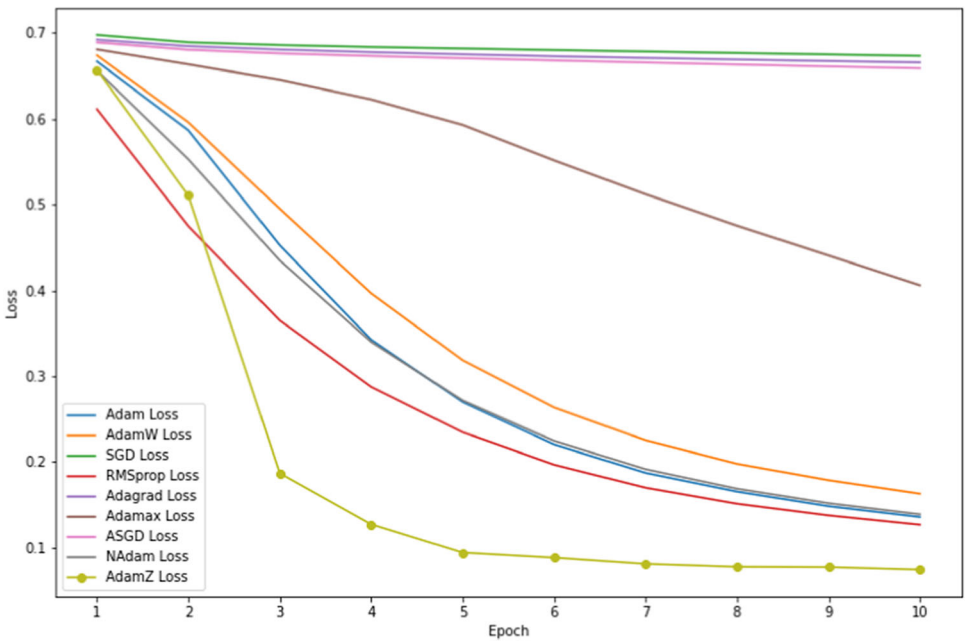

Fig. 7 The loss evolution of the optimisers for the make_circles dataset

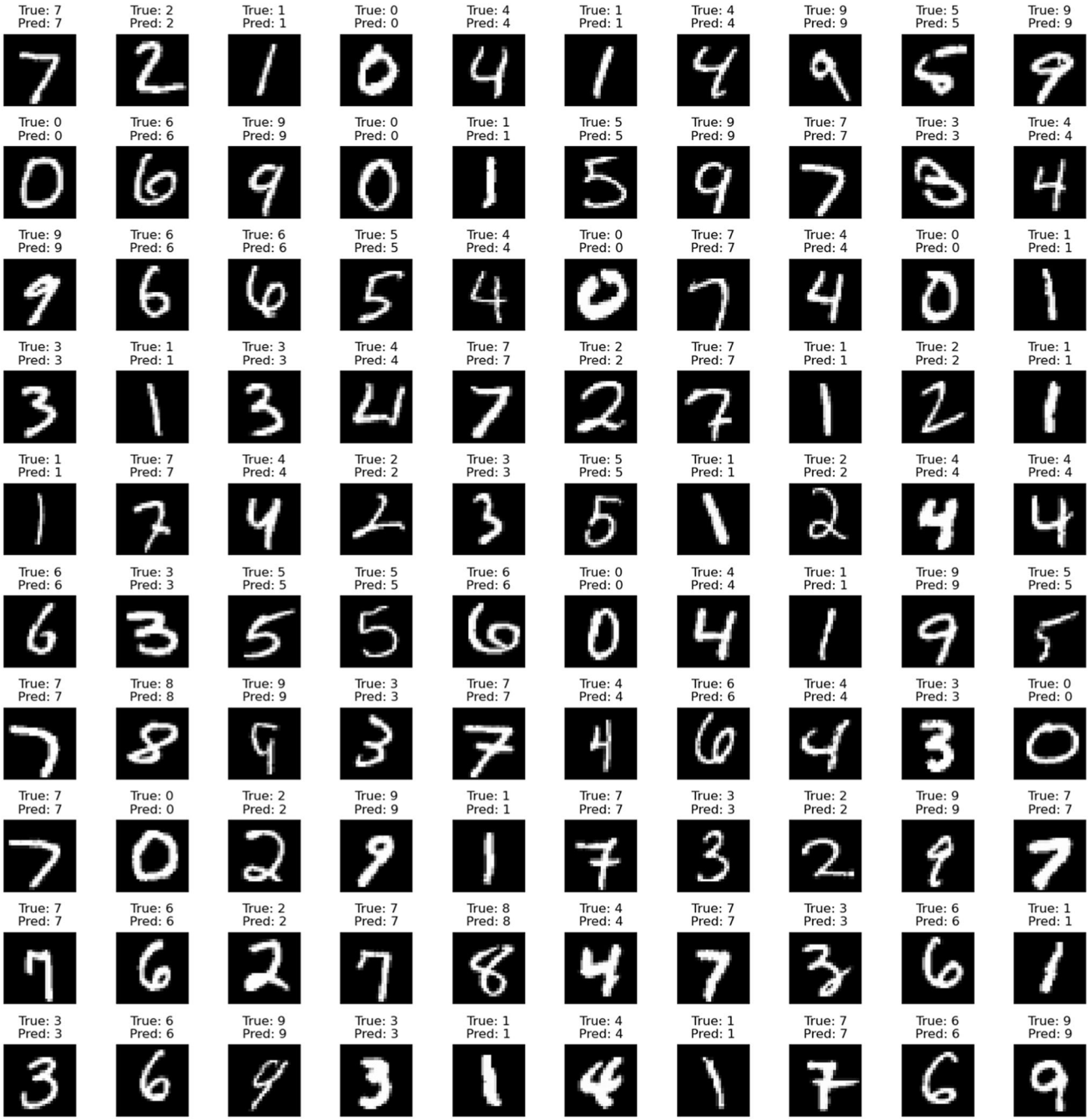

Fig. 8 Visualisation of the MNIST dataset

The MNIST dataset provides a more complex and real-world scenario to evaluate the effectiveness of various optimisers and neural network architectures. Its simplicity, yet challenging nature, makes it a standard benchmarking dataset for new algorithms in the field of machine learning.

4.3.2 Model architecture

Figure 9 illustrates the deep neural network architecture deployed with multiple layers, including a multi-head attention mechanism to handle the complexity of the MNIST dataset.

Hyperparameters were optimised for each optimiser to ensure a fair comparison.

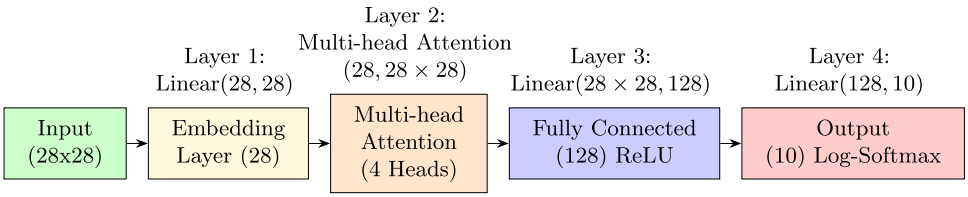

Fig. 9 Neural network architecture with Multi-head Attention for the MNIST classification task

4.3.3 Training setup

The performance of the newly developed optimiser, AdamZ, was rigorously evaluated against well-established optimisers such as Adam, Stochastic Gradient Descent (SGD), and RMSprop. This evaluation provided a comprehensive benchmark for assessing improvements in model classification accuracy, training time, and loss minimisation. The experiments spanned five epochs of training, each comprising of 1,000 steps, with 100 simulations to account for randomness in parameter initialisation. These tests were conducted on a high-performance computer cluster equipped with four A100 GPUs, ensuring robust computational support for the experiments.

4.3.4 Results

As depicted in Figure 10 and summarised in Table 4, of all the optimisers tested, AdamZ achieved the highest classification accuracy. However, it required slightly more training time, of the order of seconds, compared to the other methods. This trade-off highlights the effectiveness of AdamZ in enhancing accuracy, albeit with a marginal increase in computational time.

Additionally, confidence intervals were calculated for accuracy using the 95% confidence level, as shown in Table 5.

Looking at the training loss evolution of the optimisers in Fig. 11, it can be seen that AdamZ has minimised the loss faster than other optimisers.

The results from this experiment highlight that AdamZ achieved the highest median accuracy (95.91%), making it the best-performing optimiser on this task. Furthermore, AdamZ's confidence interval [95.86, 95.95]

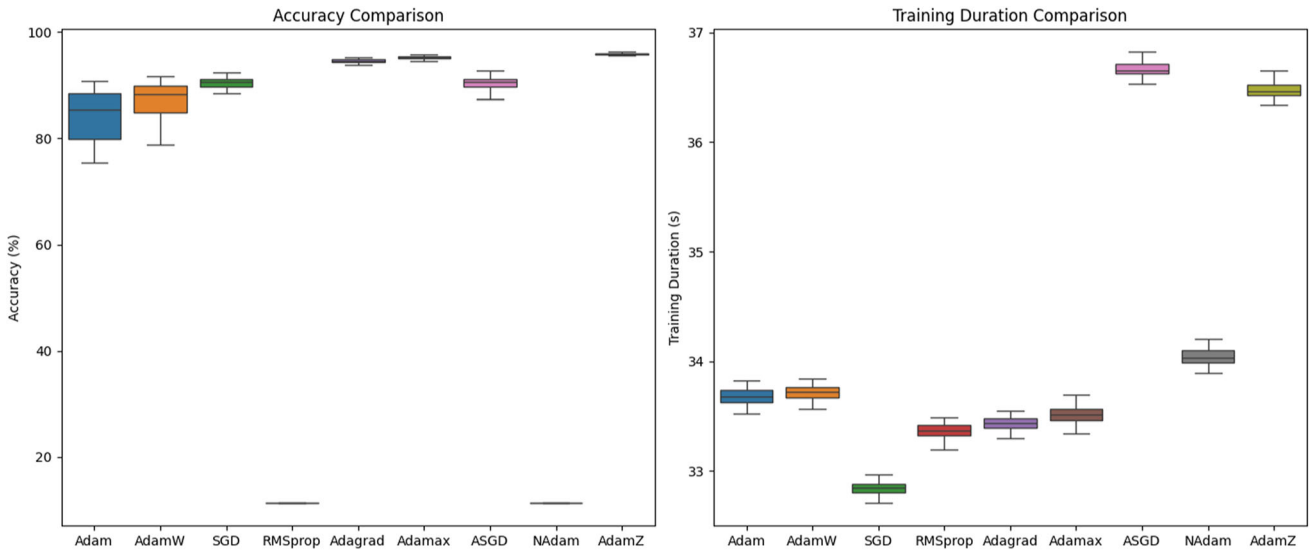

Fig. 10 Accuracy and training duration on the MNIST dataset

Table 4 Performance comparison of optimisers on the MNIST dataset

Optimiser	Accuracy			Training duration		
	Q1	Median	Q4	Q1	Median	Q4
Adam	79.91	85.42	88.41	33.62	33.68	33.73
AdamW	84.87	88.28	89.97	33.67	33.72	33.76
SGD	89.77	90.59	91.19	32.80	32.84	32.88
RMSprop	11.35	11.35	11.35	33.32	33.37	33.42
Adagrad	94.30	94.60	94.85	33.39	33.43	33.48
Adamax	95.01	95.21	95.39	33.46	33.51	33.56
ASGD	89.67	90.55	91.21	36.63	36.65	36.71
NAdam	11.35	11.35	11.36	33.99	34.03	34.10
AdamZ	95.82	95.91	96.01	36.43	36.46	36.52

Table 5 Confidence intervals for accuracy on the MNIST dataset

Optimiser	Median accuracy	95% CI lower	95% CI upper
Adam	85.42	64.98	77.03
AdamW	88.28	71.77	82.47
SGD	90.59	90.29	90.69
RMSprop	11.35	10.71	13.28
Adagrad	94.60	94.51	94.65
Adamax	95.21	95.13	95.25
ASGD	90.55	90.20	90.63
NAdam	11.35	15.00	24.42
AdamZ	95.91	95.86	95.95

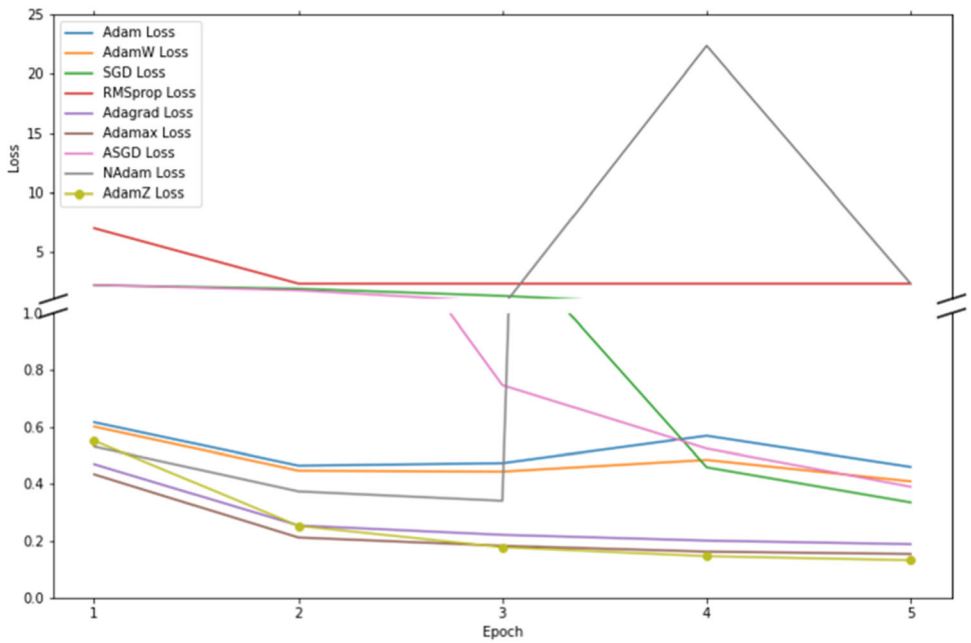

Fig. 11 Optimisers' loss evolution for the MNIST dataset

reflects its consistent performance across simulations, outperforming other widely used optimisers such as AdamW, SGD, and Adamax.

While Adamax and Adagrad demonstrated strong median accuracies of 95.21% and 94.60%, respectively, AdamZ's narrow confidence interval suggests greater reliability in performance compared to these methods. SGD

also performed well, achieving a median accuracy of 90.59% with a tight confidence interval [90.29, 90.69], indicating consistent results.

4.4 Experiment 3: CIFAR-10 dataset with a convolutional neural network (CNN)

4.4.1 Dataset description

This experiment evaluates the performance of AdamZ and other optimisers on the CIFAR-10 dataset, a widely used benchmark for image classification tasks, such as object detection and classification in autonomous vehicles, industrial automation, and medical imaging.

The CIFAR-10 dataset, introduced by Alex Krizhevsky and Geoffrey Hinton [37], is a benchmark dataset widely used for image classification tasks. It consists of 60,000 colour images divided into 10 classes, such as aeroplanes, cars, birds, and cats. Each image is of size 32x32 pixels and contains three RGB colour channels. The dataset is split into 50,000 images for training and 10,000 images for testing, making it suitable for evaluating the performance of machine learning models across diverse categories. For computational efficiency, subsets of 10,000 training samples and 2,000 testing samples were used.

Key features of the CIFAR-10 dataset include:

- *Complexity*: The dataset presents a more challenging classification task compared to simpler datasets like MNIST due to its higher dimensionality and colour information.
- *Balanced classes*: Each class is equally represented, ensuring a balanced dataset for training and testing.
- *Preprocessing*: Images are normalised to improve model convergence during training.

Figure 12 illustrates examples from the CIFAR-10 dataset, showcasing the diversity of classes and the complexity of the classification task.

4.4.2 Model architecture

The neural network architecture deployed is a Convolutional Neural Network (CNN), featuring convolutional layers for feature extraction, max-pooling for downsampling, and fully connected layers for classification. A dropout layer was added to reduce overfitting. The architecture is illustrated in Fig. 13.

4.4.3 Training setup

The model was trained for 10 epochs with a batch size of 128 across 100 simulations to account for randomness in parameter initialisation. The performance of AdamZ was compared against other optimisers, including Adam, AdamW, SGD, RMSprop, Adagrad, Adamax, ASGD, and NAdam. Metrics evaluated include classification accuracy, training duration, and loss minimisation.

4.4.4 Results

Results are summarised in Table 6 and visualised in Fig. 14.

Additionally, confidence intervals were calculated for accuracy using the 95% confidence level, as shown in Table 7.

The loss evolution for each optimiser is depicted in Figure 15, demonstrating AdamZ's superior ability to minimise loss compared to other methods.

The results from this experiment highlight that Adagrad achieved the highest median accuracy (34%), making it the best-performing optimiser on this task. However, AdamZ demonstrated significant strengths as the second-

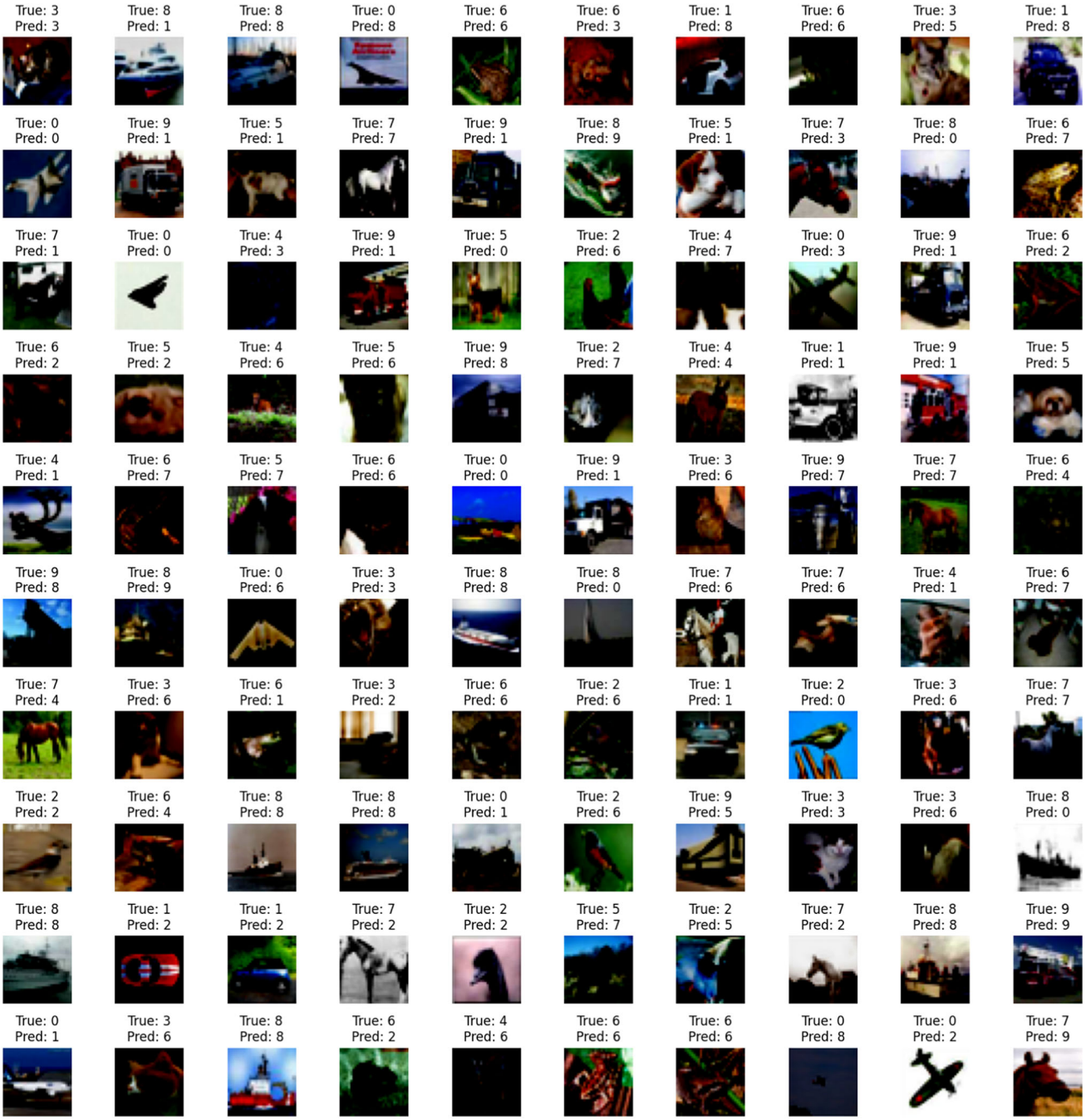

Fig. 12 Examples from the CIFAR-10 dataset. Each image is labelled with its true class, alongside the predictions made by the model. This visualisation demonstrates the diversity of the dataset and the complexity of the classification task

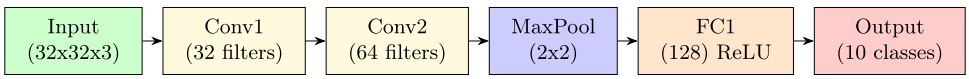

Fig. 13 CNN architecture for CIFAR-10 classification

best optimiser, achieving a median accuracy of 28%, which is notably higher than the popular Adam optimiser (8%). Furthermore, AdamZ's confidence interval [25.5, 30.5] reflects its consistent performance across simulations, outperforming other widely used optimisers such as AdamW, RMSprop, and SGD.

Table 6 Performance comparison of optimisers on the CIFAR-10 dataset

Optimiser	Accuracy			Training duration		
	Q1	Median	Q4	Q1	Median	Q4
Adam	7.0	8.0	19.5	1.152	1.157	1.165
AdamW	7.0	8.5	19.75	1.154	1.158	1.165
SGD	17.375	19.25	21.5	1.146	1.149	1.158
RMSprop	14.875	17.5	20.5	1.151	1.155	1.162
Adagrad	32.375	34.0	36.5	1.149	1.154	1.163
Adamax	9.375	17.5	26.5	1.150	1.154	1.163
ASGD	17.5	19.5	21.5	1.192	1.196	1.207
NAdam	21.375	25.75	28.0	1.155	1.160	1.168
AdamZ	25.5	28.0	30.5	1.201	1.204	1.216

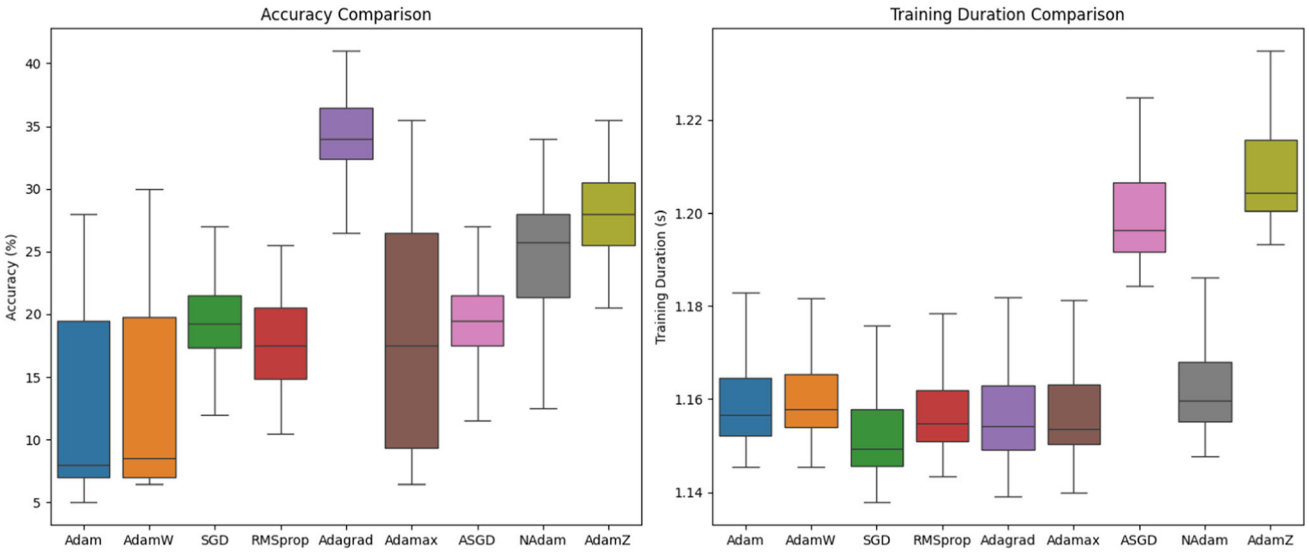

Fig. 14 Accuracy and training duration on CIFAR-10

Table 7 Confidence intervals for accuracy on the CIFAR-10 dataset

Optimiser	Median accuracy	95% CI lower	95% CI upper
Adam	8.0	10.81	13.69
AdamW	8.5	11.27	14.27
SGD	19.25	18.70	20.01
RMSprop	17.5	16.94	18.36
Adagrad	34.0	33.66	34.86
Adamax	17.5	16.36	19.83
ASGD	19.5	18.80	20.05
NAdam	25.75	22.08	24.98
AdamZ	28.0	26.64	28.50

AdamZ's dynamic learning rate adjustments contributed to its superior loss minimisation, as shown in Fig. 15. While Adagrad excelled in accuracy, AdamZ offers a balanced trade-off between accuracy and adaptability, making it a strong contender in scenarios where precision and stability are critical.

These findings reinforce AdamZ's potential as a robust optimiser capable of competing with and outperforming many established methods, including the widely adopted Adam.

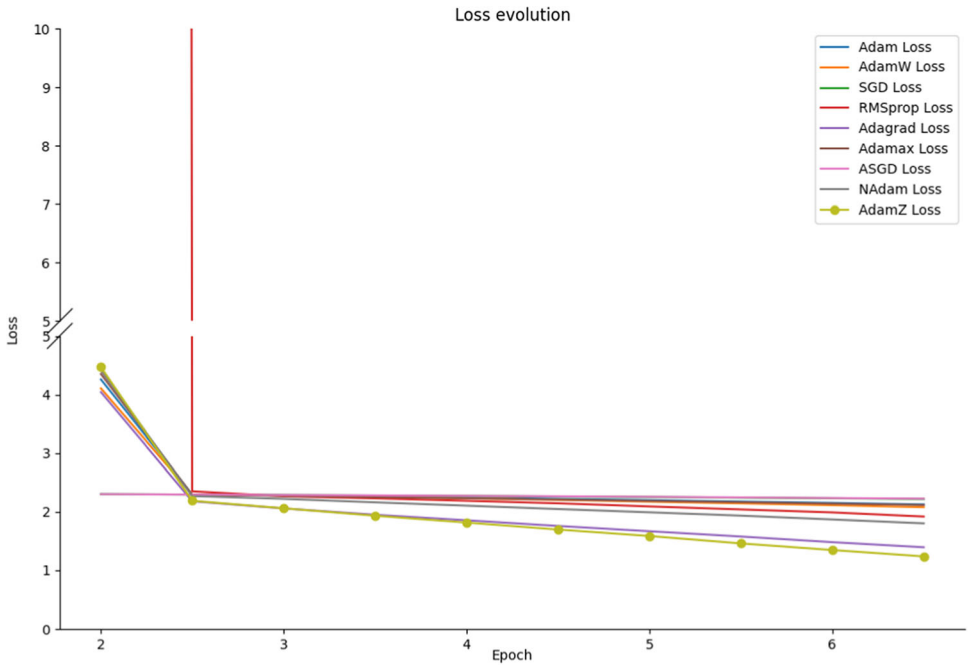

Fig. 15 Loss evolution for CIFAR-10 optimisers

4.5 Analysis of results

The experiments highlight a clear trade-off in AdamZ's performance: while it achieves the highest accuracy on MNIST and consistently excels at loss minimisation across all tasks, on the more challenging CIFAR-10 dataset AdamZ ranks as the second-best optimiser for accuracy-outperformed only by Adagrad. However, AdamZ demonstrates more stable and adaptable learning, maintaining strong performance even as task complexity increases, and continues to minimise loss effectively. This comes at the cost of slightly longer training times, reflecting the computational overhead of its dynamic learning rate adjustments. The experiments highlight a clear trade-off in AdamZ's performance: while it achieves the highest accuracy by more effectively minimising the loss, particularly on more challenging tasks such as MNIST, this comes at the cost of slightly longer training times. This is attributed to the dynamic learning rate adjustments, which are computationally more intensive but result in improved convergence. The trade-off becomes more pronounced in more complex models, such as the MNIST experiment, where the deep architecture amplifies the computational overhead. However, the scalability of AdamZ across both shallow and deep networks demonstrates its robustness, making it a viable choice for applications where accuracy is paramount. For instance, in the MNIST experiment, AdamZ achieved a median accuracy of 95.91%, outperforming other optimisers, but required approximately 2-3 s longer in training time. This trade-off is particularly beneficial for tasks that prioritise accuracy over speed, such as in cloud environments where accuracy often outweighs time constraints. For instance, AdamZ's ability to dynamically adjust learning rates reduces the number of iterations required to achieve optimal loss minimisation, which can offset the additional overhead in many scenarios.

However, AdamZ is not without limitations. Its dynamic learning rate adjustment introduces additional hyperparameters, increasing the complexity of tuning and requiring careful validation for different tasks. The slightly longer training times may be disadvantageous in settings where computational efficiency is paramount. Moreover, as observed on the CIFAR-10 dataset, AdamZ does not always outperform optimisers like Adagrad, particularly for problems with highly sparse gradients. In tasks with extreme noise or non-stationarity, inappropriate calibration of stagnation or overshooting thresholds could also lead to suboptimal adaptation.

Overall, these findings suggest that AdamZ is a promising candidate for applications requiring high accuracy and reliability in model predictions at the expense of a marginal increase in latency, which could be tolerated in

most applications. Future development of AdamZ could explore further optimisation of hyperparameters and potential enhancements in computational efficiency to reduce training time without compromising accuracy. The results from these experiments not only validate the effectiveness of AdamZ but also pave the way for its application to more diverse and challenging machine-learning tasks.

5 Conclusions and future work

5.1 Conclusions

The development and evaluation of the AdamZ optimiser have demonstrated its potential as a robust tool for enhancing neural network training. By dynamically adjusting the learning rate to limit overshooting and stagnation, AdamZ effectively improves convergence stability and model accuracy. The experimental results underscore the superior performance of this optimiser in minimising loss and achieving higher accuracy, particularly in complex datasets such as MNIST and CIFAR-10. Despite its slightly longer training times, the ability of AdamZ to maintain optimal learning rates positions it as a valuable asset in applications where precision is critical.

The comparative analysis with well-established optimisers, such as Adam, SGD, and RMSprop, demonstrated the strengths of AdamZ in navigating the intricate landscapes of neural network training. The advanced mechanisms of the optimiser for a learning rate adjustment, guided by hyperparameters such as overshoot and stagnation factors, thresholds, and patience levels, provide a dynamically responsive but tightly controlled approach that enhances its adaptability and efficiency.

Nonetheless, users should be aware that AdamZ's effectiveness depends on appropriate hyperparameter tuning and may incur higher computational cost, and it may not always be optimal for all problem types or in highly resource-constrained environments.

5.2 Applications in real-world scenarios

AdamZ's dynamic learning rate adaptation makes it highly valuable in applications requiring precision and adaptability to complex systems. In engineering, AdamZ can be applied to predictive approaches for improving manufacturing processes, such as optimising welding techniques or mitigating defects like hot cracking [38–40]. Machine learning models, including regression-based methods and neural networks, have been widely used to enhance prediction accuracy and optimise process parameters. AdamZ's ability to dynamically adjust learning rates can further refine these models, enabling more reliable predictions and improved system performance.

In finance, AdamZ offers significant advantages in applications like stock price prediction, portfolio optimisation, and risk assessment [41–43]. Its dynamic learning rate adaptation is particularly useful for handling complex market trends and correlations, allowing for more precise modelling and effective decision-making. By improving the stability and convergence of financial models, AdamZ contributes to robust strategies for managing investments and assessing risks.

Across healthcare, AdamZ can be applied to predictive models for disease diagnosis and treatment planning, where accuracy is critical. Its ability to handle complex loss landscapes makes it suitable for tasks like medical imaging analysis, enabling high-precision predictions that directly impact patient outcomes [44].

These examples highlight the versatility of AdamZ across diverse domains, showcasing its potential to address critical challenges in engineering, finance, and healthcare.

5.3 Future work

Future research will focus on several key areas to further enhance the capabilities of AdamZ. Firstly, optimising the computational efficiency of AdamZ is crucial to reduce training times without compromising optimisation performance. This might involve refining the internal update rules to minimise redundant computations, implementing more efficient memory management for moment estimates, or leveraging hardware accelerations such as mixed-precision arithmetic [45]. Additionally, exploring algorithmic modifications that enable AdamZ to better scale with very large batch sizes [46] or adapt dynamically to available computational resources could further improve its utility in practical deployments.

Another promising direction is the exploration of adaptive hyperparameter tuning. Developing methods for AdamZ to automatically adjust its own hyperparameters based on real-time feedback from the training process could reduce the need for manual intervention and improve model performance. Techniques such as online hyperparameter adaptation [47], where AdamZ monitors gradient statistics and adjusts its parameters accordingly, or meta-optimisation approaches [48] that learn optimal schedules during training, represent fruitful avenues for making the optimiser more robust and user-friendly.

Additionally, expanding the application of AdamZ to more diverse and challenging machine learning tasks will be a priority. To ensure effectiveness across different neural network architectures and domains, AdamZ could be enhanced with mechanisms to detect and adapt to domain-specific training dynamics, such as varying gradient distributions or loss landscape properties [22]. For example, incorporating curvature information or domain-aware scaling factors could help AdamZ maintain stability and convergence speed in both natural language processing and computer vision settings. Systematic benchmarking and ablation studies would be necessary to identify and address any limitations that arise in these varied contexts.

Finally, integrating AdamZ with emerging technologies, such as reinforcement learning frameworks [49] or hybrid optimisation models [50], could open new avenues for innovation. For reinforcement learning, AdamZ could be adapted to better handle non-stationary objectives and high-variance gradients, possibly by introducing mechanisms for temporal credit assignment or reward-based learning rate modulation [51]. In hybrid optimisation scenarios, AdamZ's update rules could be designed to interoperate seamlessly with other algorithms, allowing for dynamic switching or blending of optimisation strategies based on training progress or model behaviour.

In conclusion, AdamZ represents a significant advancement in optimisation techniques, offering a more responsive and effective approach to learning rate adjustment. Continued research and development along these lines will ensure its relevance and utility in the ever-evolving landscape of AI Engineering.

Acknowledgements The authors gratefully acknowledge the computing time granted by the KISSKI project.

Author contributions I.Z. led the research, conducted the experiments, and drafted the initial manuscript. A.B., J.K., and A.D. provided critical guidance in shaping the research direction, refining methodologies, and interpreting results. J.K. also facilitated access to the necessary computing resources, and A.B. secured funding to support the study. All authors contributed to reviewing and improving the manuscript, reflecting a collective effort to ensure the quality and impact of the work.

Funding This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Data availability The datasets generated and/or analysed during the current study are available from the corresponding author upon reasonable request.

Code availability The code for the AdamZ optimiser and related experiments is available at the following GitHub repository: https://github.com/izaznov/AdamZ.

Materials availability Not applicable.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval and consent to participate Not applicable.

Consent for Publication All authors have provided their consent for publication.

References

1. Robbins H, Monro S (1951) A stochastic approximation method. Ann Math Stat 22(3):400–407
2. Polyak BT, Juditsky AB (1992) Acceleration of stochastic approximation by averaging. SIAM J Control Optim 30(4):838–855
3. Duchi J, Hazan E, Singer Y (2011) Adaptive sub-gradient methods for online learning and stochastic optimization. J Mach Learn Res 12(July):2121–2159
4. Tieleman T (2012) Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA Neural Networks for Mach. Learn. 4(2):26
5. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980
6. Dozat T (2016) Incorporating Nesterov momentum into Adam. In: ICLR Workshop
7. Loshchilov, I., Hutter, F (2017) Decoupled weight decay regularisation. arXiv preprint arXiv:1711.05101
8. Cauchy A-L (1847) Méthode générale pour la résolution des systèmes d'équations simultanées. C R Hebd Seances Acad Sci 25:536–538
9. Polyak BT (1964) Some methods of speeding up the convergence of iteration methods. USSR Comput Math Math Phys 4(5):1–17
10. Bottou L (1998) Online learning and stochastic approximations, 9–42
11. Nesterov Y (1983) A method of solving a convex programming problem with convergence rate $O(1/k^2)$. Soviet Math. Doklady 27(2):372–376
12. Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res 12(7)
13. Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA Neural Networks for Mach. Learn. 4(2):26–31
14. Chen X, Chen C, Ru B, He H, Chen H, Xue L, Huang C, Levine S, Gomez A (2023) Symbolic discovery of optimization algorithms. Nature 621(7978):302–308
15. Ioannou G, Tagaris T, Stafylopatis A (2023) Adalip: an adaptive learning rate method per layer for stochastic optimization. Neural Process Lett 55(5):6311–6338
16. Zhang R, Javaheripi M, Ghodsi Z, Bleiweiss A, Koushanfar F (2023) AdaGL: Adaptive Learning for Agile Distributed Training of Gigantic GNNs. In: 2023 60th ACM/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE
17. Liu DC, Nocedal J (1989) On the limited memory bfgs method for large scale optimization. Math Program 45:503–528
18. Amari S-i (1998) Natural gradient works efficiently in learning. Neural Comput 10(2):251–276
19. Luo L, Xiong Y, Liu Y, Sun X (2019) Adaptive gradient methods with dynamic bound of learning rate. arXiv preprint arXiv:1902.09843
20. Reddi SJ, Kale S, Kumar S (2019) On the convergence of Adam and beyond. arXiv preprint arXiv:1904.09237
21. Zhang S, Mitliagkas I, Ré C (2017) Normalized direction-preserving adam. arXiv preprint arXiv:1709.04535
22. Keskar NS, Mudigere D, Nocedal J, Smelyanskiy M, Tang PTP (2017) On large-batch training for deep learning: Generalization gap and sharp minima. In: International Conference on Learning Representations (ICLR)
23. Zhang MR, Lucas J, Hinton G, Ba J (2019) Lookahead optimizer: k steps forward, 1 step back. Advances in neural information processing systems **32**
24. Loshchilov I, Hutter F (2016) Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983
25. Zaheer M, Reddi S, Sachan D, Kale S, Kumar S (2018) Adaptive methods for nonconvex optimization

26. Martens J, Grosse R (2015) Optimizing neural networks with kronecker-factored approximate curvature. In: International Conference on Machine Learning, pp. 2408–2417. PMLR
27. Gupta V, Koren T, Singer Y (2018) Shampoo: Preconditioned stochastic tensor optimization. In: International Conference on Machine Learning, pp. 1842–1850. PMLR
28. Yao Z, Gholami A, Shen S, Keutzer K, Mahoney MW (2021) Adahessian: an adaptive second order optimizer for machine learning. Proceed AAAI Confer Artific Intell 35(12):10665–10673
29. Zuo X, Zhang P, Gao S, Li H-Y, Du W-R (2023) NALA: A Nesterov Accelerated Look-Ahead optimizer for deep neural networks
30. Holland JH (1984) Genetic algorithms and adaptation. In: Adaptive Control of Ill-defined Systems, pp. 317–333
31. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks, pp. 1942–1948
32. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
33. Tapson J, Schaik A (2012) Learning the pseudoinverse solution to network weights. arXiv preprint arXiv:1207.3368
34. Rodini S (2022) Analytical derivatives of neural networks. Comput Phys Commun 270:108169
35. Azevedo BF, Rocha AMA, Pereira AI (2024) Hybrid approaches to optimization and machine learning methods: a systematic literature review. Mach Learn 113(7):4055–4097
36. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
37. Krizhevsky A, Hinton G (2010) Convolutional deep belief networks on. Unpublished manuscript 40(7), 1–9
38. Dhilip A, Nampoothiri J, Senthilkumar M, Kirubanandan N (2023) Machine learning predictive approaches for hot crack mitigation in modified tig welded aa7075 joints. Mater Manuf Processes 38(13):1650–1662
39. Annamalai D, Nampoothiri J (2024) Ultrasonic-assisted tungsten inert gas welding of inconel 625 joints to reduce hot cracking and improve microhardness: Optimization and prediction methods. J Test Eval 52(4):2515–2537
40. Nampoothiri J et al (2025) Leveraging machine learning to predict welding quality in ultrasonically assisted tig welded inconel 625 joints. Journal of Materials Engineering and Performance, 1–15
41. Zaznov I, Kunkel J, Dufour A, Badii A (2022) Predicting stock price changes based on the limit order book: a survey. Mathematics 10(8):1234
42. Zaznov I, Kunkel JM, Badii A, Dufour A (2024) The intraday dynamics predictor: a trioflow fusion of convolutional layers and gated recurrent units for high-frequency price movement forecasting. Appl Sci 14(7):2984
43. Fatouros G, Makridis G, Kotios D, Soldatos J, Filippakis M, Kyriazis D (2023) Deepvar: a framework for portfolio risk assessment leveraging probabilistic deep neural networks. Digit Finance 5(1):29–56
44. Anwar SM, Majid M, Qayyum A, Awais M, Alnowami M, Khan MK (2018) Medical image analysis using convolutional neural networks: a review. J Med Syst 42(11):226
45. Micikevicius P, Narang S, Alben J, Diamos G, Elsen E, Garcia D, Ginsburg B, Houston M, Kuchaiev O, Venkatesh G, et al (2017) Mixed precision training. arXiv preprint arXiv:1710.03740
46. Goyal P, Dollár P, Girshick R, Noordhuis P, Wesolowski L, Kyrola A, Tulloch A, Jia Y, He K (2017) Accurate, large minibatch sgd: Training imagenet in 1 hour. arXiv preprint arXiv:1706.02677
47. Baydin AG, Cornish R, Rubio DM, et al (2018) Online learning rate adaptation with hypergradient descent. ICLR
48. Andrychowicz M (2016) Learning to learn by gradient descent by gradient descent. In: NeurIPS
49. Sutton RS, Barto AG, et al (1998) Reinforcement learning: An introduction **1**(1)
50. Jaderberg M, Dalibard V, Osindero S, Czarnecki WM, Donahue J, Razavi A, Vinyals O, Green T, Dunning I, Simonyan K, et al (2017) Population based training of neural networks. arXiv preprint arXiv:1711.09846
51. Donâncio H, Barrier A, South LF, Forbes F (2024) Dynamic learning rate for deep reinforcement learning: a bandit approach. arXiv preprint arXiv:2410.12598

Authors and Affiliations

Ilia Zaznov[1] · Atta Badii[1] · Julian Kunkel[2] · Alfonso Dufour[3]

✉ Ilia Zaznov
i.zaznov@pgr.reading.ac.uk

Atta Badii
atta.badii@reading.ac.uk

https://doi.org/10.1007/s00521-025-11649-w

Julian Kunkel
julian.kunkel@gwdg.de

Alfonso Dufour
a.dufour@icmacentre.ac.uk

[1] Department of Computer Science, University of Reading, Reading, UK

[2] Department of Computer Science/GWDG, University of Göttingen, Goettingen, Germany

[3] ICMA Centre, Henley Business School, University of Reading, Reading, UK