Adam: A Method for Stochastic Optimization

Gurman Basran
Qusai Quresh
10th April 2026

Gradient descent is the foundation of nearly every modern machine learning training procedure. In its simplest form, Stochastic Gradient Descent (SGD) updates a parameter vector θ by subtracting a fixed multiple of the gradient of the loss function: $\theta_t = \theta_{t-1} - \alpha \nabla f(\theta_{t-1})$. In practice, however, vanilla SGD suffers from two fundamental problems that make it poorly suited for large-scale, high-dimensional optimization. First, a single step size α is applied to every parameter equally, even though different parameters may have gradients that differ by orders of magnitude. When the loss landscape is stretched in one direction due to high curvature, SGD zigzags back and forth across the narrow direction and makes almost no progress along the wide direction. This is formalized by the class convergence theorem (Theorem 3.4): the convergence rate of steepest descent is $(\kappa - 1)/(\kappa + 1)$, so at $\kappa = 2500$ each iteration only reduces the error by 0.1%. Second, in machine learning the true gradient is never available; only a noisy estimate $g_t \approx \nabla f(\theta_t)$ from a small random minibatch can be computed, which amplifies the zigzagging problem further. Methods like Newton's method and BFGS, studied in class, cannot help here because they require exact gradients and an $n \times n$ curvature matrix, which is impossible to store for models with millions of parameters.

The goal of this project is to implement, analyze, and experimentally evaluate the Adam optimizer (Adaptive Moment Estimation), introduced by Kingma and Ba at ICLR 2015 [1]. Adam directly addresses both problems described above by combining two ideas, momentum and adaptive learning rates, along with a bias correction mechanism that makes the algorithm well-behaved from the very first iteration. The project reproduces key experimental results from Sections 6.1 and 6.2 of the original paper, validates the implementation against PyTorch's built-in Adam optimizer, and extends the comparison to the Rosenbrock function and a hyperparameter sensitivity study. Five optimizers were implemented from scratch in NumPy: SGD, SGD with Momentum, AdaGrad, RMSProp, and Adam.

Adam synthesizes two ingredients that were developed independently in the optimization literature. The first is momentum: instead of using the raw gradient g_t directly, a running exponential moving average of past gradients is maintained, $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$, where $\beta_1 = 0.9$ by default. This smooths out the noise in individual minibatch gradient estimates and gives a steady, consistent direction of descent. The second ingredient is adaptive learning rates: a second running average tracks how volatile each parameter's gradient has been, $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$, where $\beta_2 = 0.999$ by default. Parameters with large gradients receive smaller effective steps, preventing overshooting, and parameters with small gradients receive larger steps, preventing stalling. This gives each of the n parameters its own automatic learning rate using only $O(n)$ memory, compared to the $O(n^2)$ required by BFGS for full curvature scaling. Adam can be thought of as a practical diagonal approximation of the BFGS inverse Hessian, designed for the noisy, high-dimensional setting of machine learning.

Both m_t and v_t are initialized to zero, which introduces a systematic downward bias in the early iterations. At $t = 1$ with $\beta_1 = 0.9$, the raw estimate is $m_1 = 0.1\, g_1$, which is ten times too small. Adam corrects this by dividing: $\hat{m}_t = m_t/(1 - \beta_1^t)$ and $\hat{v}_t = v_t/(1 - \beta_2^t)$. The correction is large early on and vanishes as t grows. The complete update rule is then $\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t/(\sqrt{\hat{v}_t} + \varepsilon)$, with default hyperparameters $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-8}$. The ratio $\hat{m}_t/\sqrt{\hat{v}_t}$ gives each parameter its own appropriately scaled step; the paper shows that each step is approximately bounded by α, which is what makes Adam robust even when gradients are extremely large. The theoretical convergence guarantee (Theorem 4.1 in [1]) states that under convexity and bounded gradients, the average regret satisfies $R(T)/T = O(1/\sqrt{T})$, matching the best known rate for

online convex optimization.

All five optimizers were implemented from scratch in Python using only NumPy. The Adam class follows Algorithm 1 from the paper line by line: the timestep counter increments, the biased moment estimates m_t and v_t are updated, the bias corrections $\hat{m}_t$ and $\hat{v}_t$ are computed, and finally the parameter update is applied. The only difference from the paper's pseudocode is NumPy vectorization, so all 7,850 parameters of the logistic regression model (and all 101,770 parameters of the MLP) update simultaneously in a single array operation. The model for the MNIST experiments uses softmax output and cross-entropy loss, with gradients computed in closed form as $dW = X^T(\hat{y} - y)/b$ where b is the batch size. The training loop uses minibatches of size 128 randomly shuffled at each epoch.

Five experiments were conducted to evaluate Adam's performance and validate the implementation. The first experiment reproduces Section 6.1 of Kingma and Ba [1]. All five optimizers were trained on the full MNIST dataset (60,000 images, 784 features, 10 digit classes) for 40 epochs with minibatch size 128, starting from the same random initial weights. Adam achieved a test accuracy of 92.69% and a final training loss of 0.258, compared to 91.58% and 0.321 for plain SGD. AdaGrad reached the highest accuracy at 92.72% with a loss of 0.264, and RMSProp achieved 92.65% and 0.260. SGD with Momentum came in at 92.36% and 0.274. All adaptive methods outperform plain SGD, consistent with the paper's reported trends, though the differences between Adam, AdaGrad, and RMSProp are small because logistic regression is a convex problem. The gap is visible most clearly on the log-scale loss curve, where the adaptive methods descend to a lower loss plateau than SGD within the first few epochs.

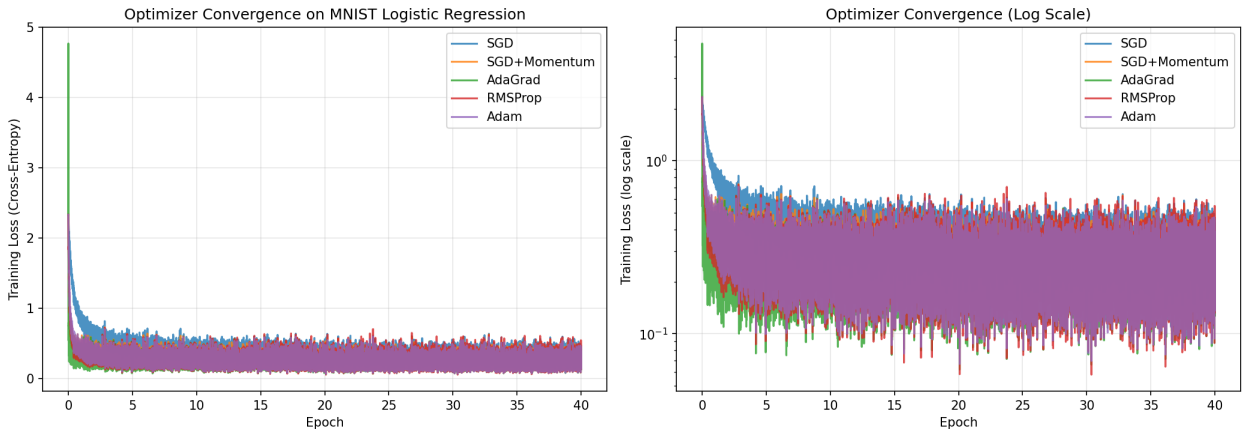

Figure 1: Experiment 1: training loss vs epoch on MNIST logistic regression, linear scale (left) and log scale (right). All five optimizers start from the same random initial weights. The log scale reveals that adaptive methods converge to a lower loss plateau than plain SGD.

The second experiment validates that the from-scratch NumPy implementation is correct by comparing it against PyTorch's `torch.optim.Adam` on identical data, with the same initial weights, the same minibatch ordering, and the same hyperparameters. Over 4,690 iterations (10 epochs, 469 batches each), the maximum absolute difference in loss between the two implementations was 6.34×10^{-7}, and the mean difference was 1.24×10^{-9}. The two loss curves are visually indistinguishable at both full scale and when zoomed in on the first 500 iterations. The tiny residual difference is attributable to floating-point rounding between NumPy and PyTorch and is not an

error in the implementation. PyTorch's source code was checked directly to confirm it implements the same version of Algorithm 1. This experiment provides strong evidence that all subsequent experimental results can be trusted.

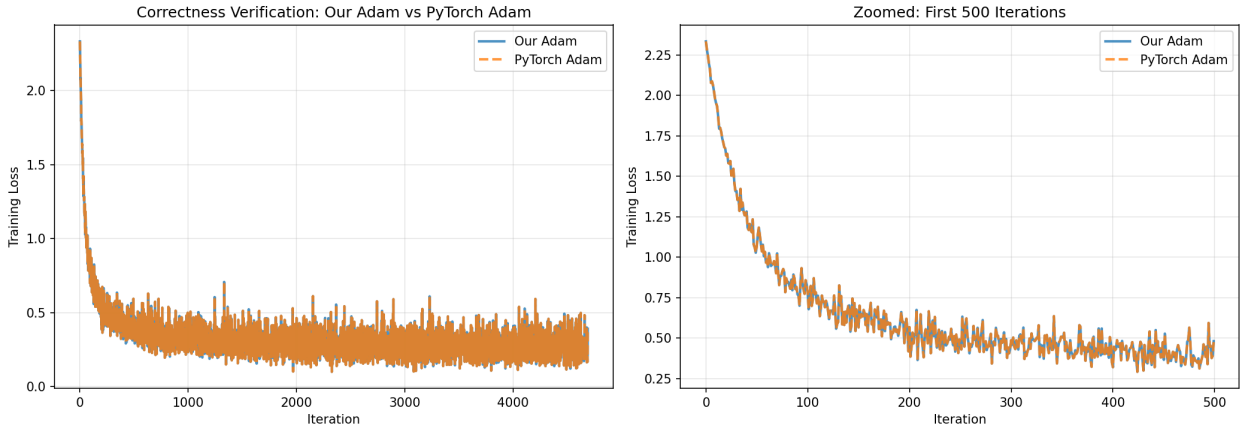

Figure 2: Experiment 2: our NumPy Adam vs PyTorch `torch.optim.Adam` over 4,690 iterations. Full run (left) and zoomed view of the first 500 iterations (right). The two curves are visually indistinguishable at both scales.

The third experiment connects the project back to material from the course by testing all five optimizers on the Rosenbrock function $f(x, y) = (1-x)^2+100(y-x^2)^2$, a deterministic benchmark with a narrow curved valley and a condition number of approximately $\kappa \approx 2500$ near the minimum at $(1, 1)$. This experiment was not part of the original paper; it was added to examine whether Adam's advantages carry over to a setting where exact gradients are available. All optimizers were started from $(-1, 1)$. SGD reached a final value of $f = 1.47$, which matches the class prediction that steepest descent barely converges at $\kappa = 2500$ because the per-step error reduction is $(\kappa - 1)/(\kappa + 1) \approx 0.999$. SGD with Momentum reached $f = 0.000069$ and AdaGrad reached $f = 0.000002$, both effectively finding the minimum. RMSProp reached $f = 0.000225$, also very close. Adam, however, stalled at $f = 0.043$, which is better than plain SGD but considerably worse than the other methods. This is an important negative result: Adam was designed for the stochastic gradient regime where minibatch noise is the dominant difficulty, and on deterministic problems with exact gradients the adaptive second-moment estimate does not provide the same advantage.

The fourth experiment investigates which of Adam's hyperparameters matters most, reproducing the analysis in Section 6.4 of the paper. Each of α, β_1, and β_2 was varied one at a time while the other two were held at their default values, training logistic regression on MNIST for 20 epochs each. Varying α from 0.0001 to 0.005 produced final losses ranging from 0.267 to 0.209, a spread of 0.058. By contrast, varying β_1 from 0 to 0.99 only changed the final loss by 0.005, and varying β_2 from 0.9 to 0.9999 changed it by only 0.008. The best result was obtained at $\alpha = 0.005$, which achieved a final loss of 0.209. These results confirm the paper's claim in Section 6.4 that α is the only hyperparameter requiring careful selection, while β_1 and β_2 are robust across a wide range. In practice this means users need only tune a single hyperparameter, which is one of Adam's most useful properties.

The fifth experiment reproduces Section 6.2 of the paper by scaling up from logistic regression to a two-layer neural network. The architecture is a multi-layer perceptron with 784 input units,

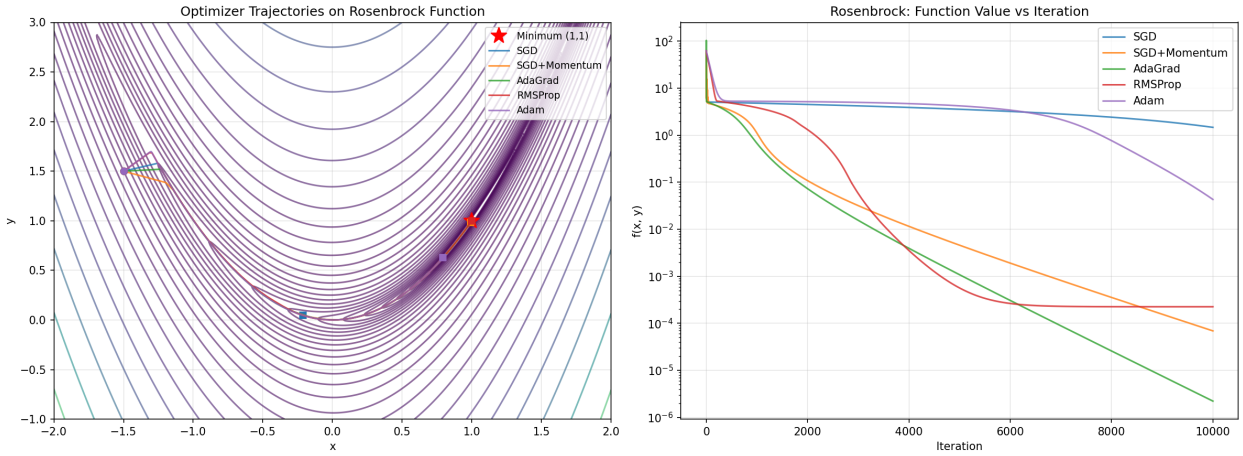

Figure 3: Experiment 3: Rosenbrock function. Left: optimizer trajectories overlaid on a contour map, starting from $(-1, 1)$ (black dot) toward the minimum at $(1, 1)$ (red star). SGD barely moves while all others follow the curved valley. Right: function value vs iteration on a log scale. SGD+Momentum and AdaGrad reach below 10^{-4} while Adam stalls at $f \approx 0.043$.

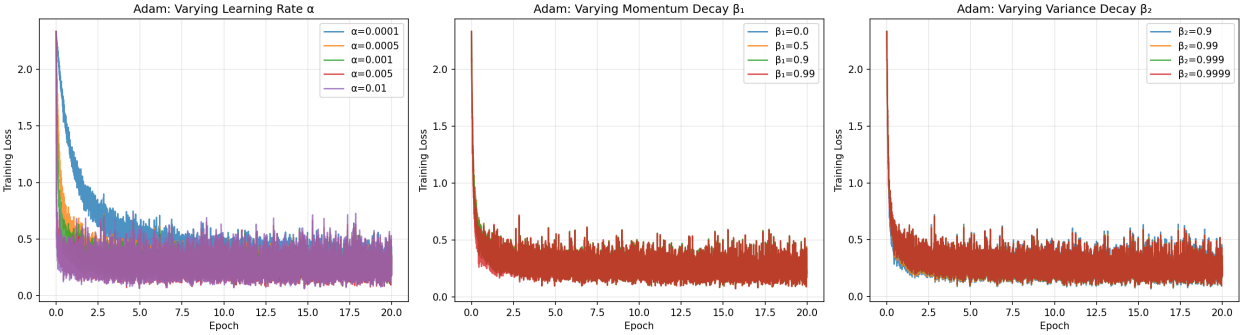

Figure 4: Experiment 4: training loss after 20 epochs while varying α (left), β_1 (centre), and β_2 (right) one at a time, keeping the other two at their default values. The loss curves spread far apart across values of α while the β_1 and β_2 curves cluster tightly together.

128 hidden units with ReLU activations, and 10 softmax output units (101,770 total parameters), trained for 20 epochs with minibatch size 128 and He initialization, with backpropagation implemented from scratch. With roughly thirteen times as many parameters as the logistic regression model and a non-convex loss landscape, this is the setting where adaptive step sizes are expected to provide the largest advantage. Adam achieved a test accuracy of 97.97% versus 93.80% for plain SGD, a gap of over four percentage points. RMSProp was close behind at 97.92%, SGD with Momentum reached 97.65%, and AdaGrad achieved 97.45%. The gap between Adam and SGD is more than four times larger here than in Experiment 1, confirming that Adam's benefits grow substantially when moving from convex to non-convex problems and from small to large models. The near-identical performance of Adam and RMSProp is consistent with the relationship between the two methods: Adam is RMSProp plus momentum plus bias correction, so the marginal contribution of those additions is small when gradient noise is already moderate.

To conclude, these results show that Adam consistently outperforms plain SGD on the MNIST classification tasks used in the original paper, with the advantage growing from roughly

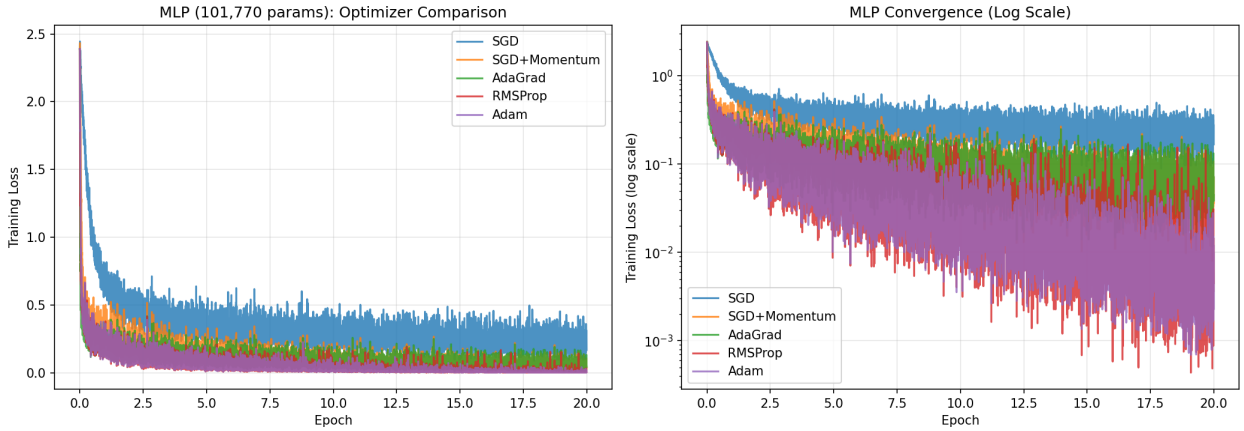

Figure 5: Experiment 5: training loss vs epoch for the two-layer MLP on MNIST, linear scale (left) and log scale (right). Adam and RMSProp descend fastest and separate clearly from SGD; on the log scale the gap is visible from the very first epoch.

one percentage point on logistic regression to over four on the neural network. The correctness of the implementation was confirmed to within 6.34×10^{-7} of PyTorch. The hyperparameter sensitivity study confirmed that only α requires tuning. The Rosenbrock experiment revealed that Adam's advantages are specific to the stochastic regime it was designed for, as simpler methods can outperform it when exact gradients are available. Overall the experiments demonstrate that Adam occupies a principled middle ground between the simplicity of gradient descent and the curvature exploitation of BFGS, achieving near-BFGS-quality adaptive scaling at $O(n)$ memory, which is what makes it the default optimizer in modern machine learning.

Stepping back, Adam is best understood as a natural evolution of the ideas covered in this course. From Chapter 2, every iterative optimizer follows the same structure $\theta_t = \theta_{t-1} + \alpha_t p_t$, and algorithms differ only in how they choose the search direction and step size. Gradient descent uses $p_k = -\nabla f$, Newton's method uses $p_k = -\nabla^2 f^{-1} \nabla f$, and Adam's direction sits between the two: a smoothed, per-parameter scaled gradient that acts as a diagonal approximation of the Newton step at $O(n)$ cost rather than $O(n^2)$. The bounded step size $|\Delta\theta_t| \lesssim \alpha$ is directly analogous to the trust-region idea from Chapter 2, where the radius limits how far each iteration moves when the local model may be inaccurate. From Chapter 1, the convergence guarantee rests on the first-order optimality condition $\nabla f(\theta^*) = 0$, and Theorem 4 of Chapter 1 ensures that for convex f any such stationary point is a global minimizer, which is exactly why the regret bound requires convexity. Adam is not a departure from the classical theory but a direct application of it to the high-dimensional, noisy setting of machine learning.

References

[1] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR 2015)*. arXiv:1412.6980. https://arxiv.org/abs/1412.6980

[2] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv:1609.04747.

[3] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*, 2nd ed. Springer.

[4] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, 2121–2159.

[5] Hinton, G., Srivastava, N., and Swersky, K. (2012). Neural networks for machine learning: Lecture 6a. Unpublished lecture notes, University of Toronto.

[6] https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

[7] https://gbasran.github.io/ADAM-Web-Presentation/