Neural Network Optimization and Regularization

Nikhil Solanki

March 2026

1 Introduction

Modern neural network performance optimization is an often long sought after achievement in modern day implementations. Through the various kinds of neural networks to their wide range of use cases, achieving optimal performance is crucial in many situations today. In many machine learning workflows, performance differences arise from subtle training decisions, such as hyperparameter tuning and optimization schedules, instead of the model's structure itself.

This report investigates the role of optimization and regularization in neural network training. More specifically, this is conducted using two supervised learning tasks: binary classification on the Adult Income dataset, and multiclass classification on the Wine Quality dataset. Using a PyTorch neural network architecture, I analyzed how different training strategies can impact the model's behavior under various experimental budgets. The study examines three components of the training process: randomized optimization methods applied to the final layers of the neural network, ablation experiments that isolate key components of the Adam optimizer, and an evaluation of regularization techniques designed to improve the model's generalization. Through systematic experimentation and budget-controlled comparisons, this work helps to identify important relationships between training decisions and model outcomes. By analyzing the convergence and generalization behavior, we can derive practical insights into when different optimization strategies or regularization techniques should be prioritized in building machine learning systems.

2 Exploratory Data Analysis and Hypotheses

This report uses the same preprocessing pipeline, dataset splits, and backbone architecture established in my previous work to ensure fair comparisons. Model selection and hyperparameter tuning were performed using the training and validation sets only, while the held-out test set was used a single time at the end of each experiment to report final performance. The same 70-30 train-validation split was used for the PyTorch MLP, due to the additional complexity of implementing cross-validation in PyTorch. To ensure reproducibility and fair optimizer comparisons, all experiments used fixed random seeds (66, 820, 9999) and identical data splits were maintained across runs. The PyTorch SL backbone architecture remained fixed throughout the study. Only in Part 3 experiments (regularization study) were additional regularization components introduced (ex: weight decay, dropout, input noise, or label smoothing), which are explicitly documented in that section of the code. EDA is shortened for brevity.

The neural networks were implemented on the Adult Income (Census Income) dataset and Wine Quality dataset (red + white). The Adult dataset is a binary classification problem, consisting of income levels for people $\leq 50k$ or $> 50k$. The dataset contains over 45,000 samples with a mix of categorical and numerical features, exhibiting moderate class imbalance ($\sim$ 3:1). The target is *income*, a binary and imbalanced variable. This motivated the use of preprocessing techniques like one-hot encoding and feature scaling (standardization) to ensure compatibility for all models.

The Wine dataset represents a multiclass classification problem, with the target of classifying wines based on their *quality* ratings. It consists of roughly 6500 entries, of solely numerical data describing aspects of the wine. The class distribution is favored towards mid-quality ratings, where models must distinguish subtle differences between neighboring quality levels.

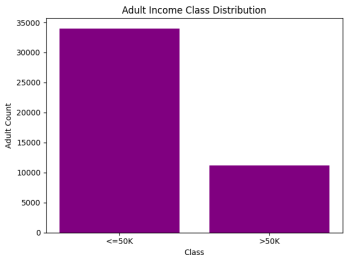
Based on these dataset characteristics and prior baseline results, the following hypotheses are proposed for the optimization experiments in this report:

(Adult Dataset)–**Hypothesis**: For the Adult dataset, I hypothesize that momentum-based and adaptive gradient methods will improve the convergence speed and validation performance compared to vanilla SGD. Randomized optimization applied to the final layers of the neural networks should marginally improve validation performance but will likely not outperform a well-tuned Adam baseline. Lastly, improvements in the final test F1 relative to the baseline should solely depend on the optimal regularization hyperparameter choices rather than optimizer selection alone.

(Wine Dataset)–**Hypothesis**: For the Wine dataset, I hypothesize that randomized optimization algorithms (RHC, SA, GA) applied to the final layers of the neural networks will exhibit more variability between seeds than on the Adult dataset due to the smaller dataset size. These methods will likely improve validation performance and consistently outperform the baseline on test.

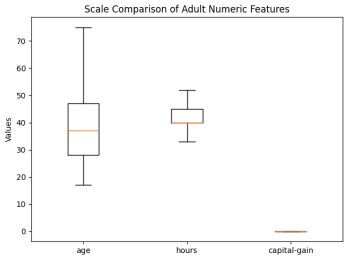
3 Randomized Optimization

Randomized Optimization (RO) methods search the parameter space using stochastic operators rather than gradient information. In this project, three common RO algorithms were used: Random Hill Climbing (RHC), Simulated Annealing (SA), and Genetic Algorithms (GA). These algorithms iteratively explore prospective candidate solutions through randomized perturbations, probabilistic acceptance criterion, or population-based evolution/acceptance, allowing them to escape local minima which may hinder the deterministic optimization methods. Unlike traditional gradient-based optimizers, RO methods solely rely on repeated evaluations of an objective function to guide the search towards better-performing solutions. The trainable parameters of the final

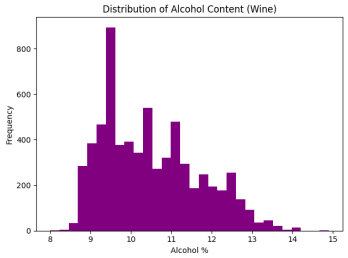


(a) Depicts large class imbalance on Adult

(b) Most observations concentrated in middle quality ratings (3–5), extreme classes (1 and 8) are rare.



(c) Shows substantial scale differences, indicating the need for feature scaling for ML models



(d) Shows distribution of alcohol content centered around 9–12%, indicating moderate spread

Figure 1: EDA on Adult and Wine Datasets.

layers (weights and biases) were flattened into a single one-dimensional vector representation. This allowed the RO algorithms to operate on the neural network parameters as a unified search space while preserving the ability to reconstruct the original layer shapes during evaluation.

3.1 Algorithm Configuration

For fair comparison, all randomized optimization methods were configured with fixed operator settings across datasets and seeds. Random Hill Climbing (RHC) used Gaussian perturbations with step size of 0.20, patience of 200, and 10 restarts.

Simulated Annealing (SA) used the same base perturbation scale with 10 restarts, initial temperature $T_0 = 2.0$, and a cooling schedule $T_t = T_0/(t+1)$, with a decaying perturbation scale $\sigma_t = max(0.01, 0.20 * 0.9985^t)$.

The Genetic Algorithm (GA) used a population size of 16, tournament selection with $k = 4$, arithmetic crossover, mutation rate of 0.2, mutation scale of 0.20, initialization scale of 0.25, elitism with 1 elite preserved per generation, and 1 restart. These settings were held constant across all runs and seeds. Mutation was applied independently to each parameter with a probability of 0.2, where perturbed weights were updated using Gaussian noise. The mutation scale was gradually reduced over the course of optimization, starting from $\sigma = 0.20$ and decaying as a function of optimization progress, with a lower bound of $\sigma = 0.03$. This schedule enables broader exploration early in the search and finer-grained refinement later.

3.2 Adult Dataset

3.2.1 Random Hill Climbing (RHC)

The Random Hill Climbing algorithm iteratively searches for improved solutions within the local neighborhood of the current state. It does this by proposing random perturbations to the current parameter vector and accepts a new solution only if it reduces the objective value via the objective function. In my implementation, neighboring solutions are generated by adding Gaussian noise to the weight vector. The algorithm continues until no improvement is observed for a set number of iterations (*patience*) or an evaluation budget is exhausted. Random restarts are employed to reduce the sensitivity to the initial state and encourage the exploration of different regions of the search space, to prevent sticking to a poor local minima.

Across all three seeds (66, 820, 9999), we can see the blue curve (RHC) decreasing as the number of function evaluation increases. This is indicative of the model improving its ability to generalize on unseen data, resulting in the "best-so-far validation loss" decreasing. RHC rapidly improved the objective during the first few hundred function evaluations (200-600) before plateauing. This behavior is indicative of the RHC algorithm performing local search in a smooth neighborhood. As a result of only optimizing the last $k = 2$ layers, the search space is relatively small and smooth. This makes it easy for the RHC algorithm to find a good basin quickly (locating strong local minima), upon which is stays indefinitely (leading to early convergence in most runs).

3.2.2 Simulated Annealing (SA)

The Simulated Annealing (SA) algorithm is inspired by the physical process of heating and cooling in metallurgy. It improves upon hill climbing by occasionally accepting worse solutions according to a "temperature"-based probability rule (Metropolis criterion, to escape local minima). In the implementation, neighboring weight vectors are generated using Gaussian perturbations with a gradually decreasing step size. The "temperature" follows a cooling schedule that reduces the likelihood of accepting worse solutions over time. This allows the algorithm to explore the search space early in the optimization and focus on refinement as the "temperature" decreases.

In comparison to RHC, SA shows slower initial convergence but continues to reduce validation loss over a larger portion of the evaluation budget. By using the Metropolis acceptance rule, worse solutions are accepted early on while the "temperature" is high. As the "temperature" decreases, the search becomes greedy. This is indicative of SA's ability to escape local minima that traps RHC. Across all three seeds, SA demonstrated stable convergence behavior and often approached similar final validation losses to RHC, though typically requiring more evaluations.

3.2.3 Genetic Algorithm (GA)

Genetic Algorithms (GA) are population-based optimization methods that evolve a set of candidate solutions through selection, crossover, and mutation. In each new generation, candidate weight vectors are evaluated and the best individual (children) are selected using a tournament selection. New solutions are produced by combining parents through arithmetic crossover, where we take a weighted average of the parents using a random interpolation factor between 0 and 1, and stochastic mutations. This implementation also enables elitism, where the best children are preserved between generations while mutation noise gradually decreases as search

continues. This enables the algorithm to balance exploration and refinement well over time.

In comparison to the previous two optimization methods, GA shows very gradual improvement over time. This is likely due to the behavior of GA in a low dimensional space (last 2 years of network), evaluating the entire population for each generation. With a 2000 evaluation budget, a lot of the evaluations succeed this and results in a population assessment. This leaves fewer opportunities for iterative improvement compared to RHC and SA. A large portion of the fixed evaluation budget is spent assessing individuals rather than iteratively refining a single solution. This leads to GA offering the slowest convergence and often the worst final validation loss within the same evaluation budget.

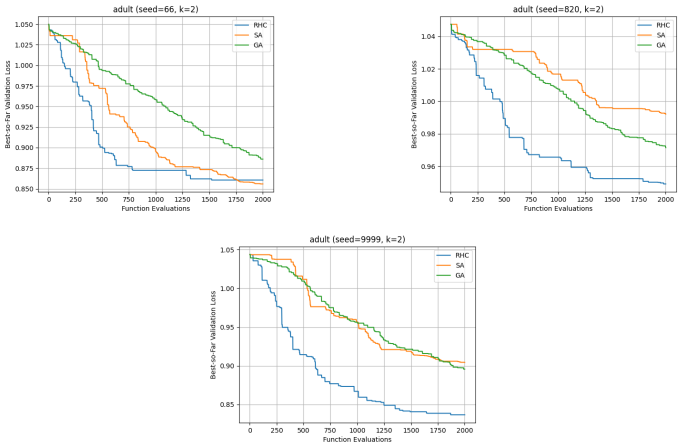


Figure 2: RHC, SA, GA validation loss across 3 seeds (Adult)

3.3 Wine Dataset

3.3.1 Random Hill Climbing (RHC)

On the Wine dataset, RHC demonstrates rapid early improvements in validation loss across all three seeds. In each run, the algorithm quickly reduced the loss within the first 100-200 function evaluations before converging towards a plateau (around 200-400 evaluations). After this, performance stabilizes and does not improve. Compared to the Adult dataset, convergence occurred earlier and exhibited greater variability across seeds. This behavior is likely due to the smaller size of the Wine dataset, which can produce a noisier validation landscape and an increased sensitivity to the initialization. As a result, the quality of the final solution varies depending on which local basin the algorithm settles in during the search.

3.3.2 Simulated Annealing (SA)

On the Wine dataset, SA demonstrates slower but more consistent improvements in validation loss compared to RHC. It also achieves rapid early reductions in loss (first 100-200 evaluations), but SA continues to improve throughout the evaluation budget due to its probabilistic acceptance of worse solutions early on in the search. In many regions where RHC stops improving, SA continues to reduce loss. This behavior allowed the algorithm to escape local minima that trapped RHC and discover lower-loss regions of the parameter space. Across all three seeds, SA frequently achieved the lowest final validation loss among the RO methods. This suggests that its exploration capabilities are superior in optimizing on smaller, more irregular datasets that contain more local minima.

3.3.3 Genetic Algorithms (GA)

Unlike RHC and SA, GA does not iteratively refine a single solution. It instead evaluates a population of candidate solutions, selects parents, and produces new children through crossover and mutation. This subsequently means that many evaluations are spent evaluating the population rather than improving one solution, which slows early progress. As a result on the Wine dataset, GA exhibited the slowest initial improvements among the RO methods. More specifically, validation loss decreased more gradually compared to RHC and SA. However, GA continued to improve steadily for the entire evaluation budget, occasionally nearing the performance of the other algorithms. The final validation loss of GA varied across all three seeds for this dataset, reflecting the stochastic nature of population initialization and mutation. As a result of this, the search can inevitably lean towards different regions of the parameter space for each run.

Overall, the Wine dataset highlights the importance of exploration in randomized optimization. While RHC quickly converges to a local minima, SA benefits from a probabilistic exploration. GA provides a broader population-based search that improves steadily but eventually requires a higher evaluation budget to reach comparable performance.

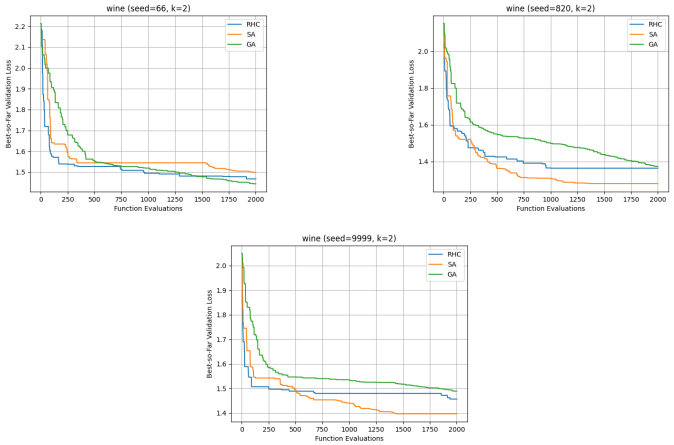


Figure 3: RHC, SA, GA validation loss across 3 seeds (Wine)

Overall, the randomized optimization methods highlight clear differences in how each algorithm explores the parameter space. RHC consistently achieved the quickest early improvements but often plateaued once it found a local minima basin. SA improved more gradually but frequently achieved the lowest final validation loss due to its ability to escape local minima through its probabilistic acceptance of worse solutions. The GA showed the slowest convergence because of the inherent overhead of population-based evaluation, yet it still steadily improved throughout the evaluation budget. The distinction in RO behavior was more pronounced in the Wine dataset, where a noisier landscape increased the model's sensitivity to initialization, enabling exploration-based methods like SA to benefit most.

Compared to the baseline test F1 of 0.6815, all randomized optimization methods performed substantially worse. RHC,

SA, and GA achieved F1 scores around 0.55, representing a decrease of roughly 0.13 in F1 relative to the gradient-based baseline.

Table 1: Randomized Optimization results compared to baseline (Adult dataset)

Method	Test Loss	Test Accuracy	Test F1	ΔF1 vs Base
Baseline	0.5940	0.8019	**0.6815**	–
Random Hill Climbing (RHC)	0.8758	0.7155	0.5455	-0.1361
Simulated Annealing (SA)	0.8612	0.7146	0.5477	-0.1339
Genetic Algorithm (GA)	0.8927	0.7146	0.5481	-0.1334

3.4 Compute Comparison Across Methods

Method	Function Evals	Gradient Evals	Test F1
RHC	2000	0	0.5455
SA	2000	0	0.5477
GA	2000	0	0.5481

Table 2: Compute accounting for randomized optimization on Adult. All RO methods were run under the same evaluation budget.

For all RO experiments, compute was measured in function evaluations, since these methods do not use gradient information; each algorithm was run with a fixed budget of 2000 function evaluations. Under comparable compute budgets, gradient-based methods achieve higher test performance with fewer effective evaluations compared to randomized optimization methods. This indicates a greater efficiency in navigating the parameter space.

The differing behavior between Adult and Wine can be attributed to dataset characteristics. The Adult dataset's larger sample size and higher-dimensional feature space produce more stable validation estimates and a smoother effective optimization landscape for the final-layer parameter search. This enables RHC to quickly converge to strong local minima. In contrast, the smaller Wine dataset introduces higher variance in validation estimates and a more irregular loss surface, making the search more sensitive to initialization and increasing the likelihood of poor local minima. As a result, exploration-based methods such as SA benefit from their ability to escape local minima, leading to improved relative performance (lowest test loss, as seen in Table 1) and higher variability across seeds compared to Adult.

4 Adam Ablations

In addition to randomized optimization methods, gradient-based optimizers were evaluated for training a neural network. Gradient-based methods use gradient information from the loss function to iteratively update the model's parameters, whereas RO relied on searching the parameter space through stochastic perturbations. This section analyzes the performance of several gradient-based optimizers: Stochastic Gradient Descent (SGD), SGD with momentum/Nesterov momentum, Adam, and AdamW. They were compared with varying hyperparameter adjustments in terms of their convergence behavior, stability across seeds, and their final performance on both datasets.

A fixed validation-loss threshold ℓ was defined based on the Adam baseline convergence behavior. Specifically, ℓ was set as:

$$\ell = best + 0.25 \cdot (start - best),$$

where *start* is the initial validation loss and *best* is the minimum validation loss achieved during a calibration run with Adam. This threshold represents a point 25% of the way from the optimal validation loss toward the initial loss, allowing consistent comparison of convergence speed across optimizers. For each optimizer, we measure the number of steps required to reach ℓ. Runs that fail to reach ℓ within the compute budget are recorded as non-convergent.

4.1 SGD-Based Methods

SGD (no momentum), SGD + momentum, SGD + Nesterov Momentum

SGD and its momentum-based variants were evaluated on the Adult dataset to examine the differences in convergence speed, stability, and final performance. Across all three seeds, SGD without momentum consistently required the largest number of updates to reach the fixed validation-loss threshold (ℓ). Both momentum-based SGD methods significantly improved convergence speed. For example, in seed 9999, vanilla SGD required approximately 1100 updates to reach the threshold, while SGD with momentum reached the same threshold in about 300 updates. Similar trends were observed in the remaining seeds as well, indicating that incorporating momentum enables the optimizer to move more efficiently across the loss landscape.

Nesterov momentum also improved convergence speed relative to vanilla SGD, but portrayed comparable performance to standard momentum. This implies that momentum-based optimizers generally trend towards similar behavior, suggesting that the primary advantage of them is to improve optimization efficiency rather than outperforming one another.

For all three seeds, each optimizer reached the target validation-loss threshold (ℓ) in comparable time, showing stable training behavior on the Adult dataset. However, validation-loss curves reveal clear differences in the optimization dynamics. Vanilla SGD initially reduces validation loss quickly and then stabilizes near its optima, whereas momentum-based methods exhibit faster early improvements but tend to become less stable as time progresses. After reaching a similar validation-loss region, the curves for SGD with momentum and Nesterov momentum gradually increase hinting at an increase risk of overfitting. This pattern is consistent with larger train-validation gaps observed for the momentum-based optimizers. Despite these differences, all optimizers converge to similar validation F1 scores (approximately 0.55), but momentum-based methods display higher variability.

4.2 Adam Ablation Study

Adam (baseline), Adam with $\beta_1 = 0$, AdamW

4.2.1 Hyperparameter Sensitivity

The Adam optimizer is a widely used adaptive gradient optimization algorithm, combining concepts from both momentum and RMSProp. It works by maintaining an exponential moving average of both the first moment (the gradient) and

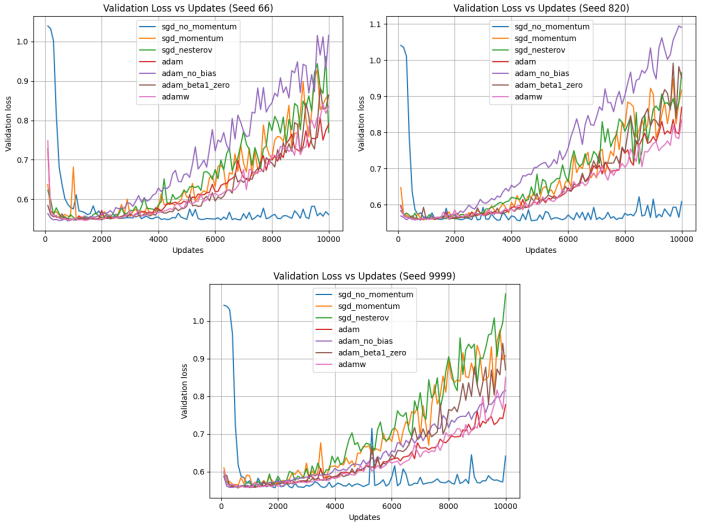


Figure 4: Validation Loss vs Updates for all optimizers

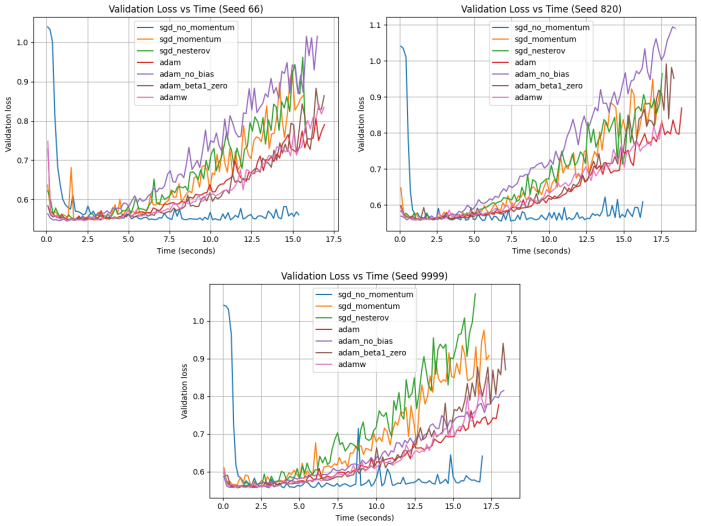


Figure 5: Validation Loss vs Time for all optimizers

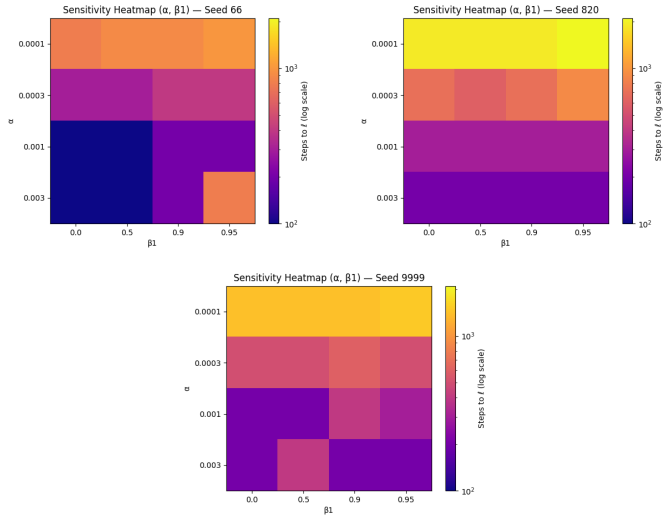


Figure 6: β_1-sweep Sensitivity Heatmap

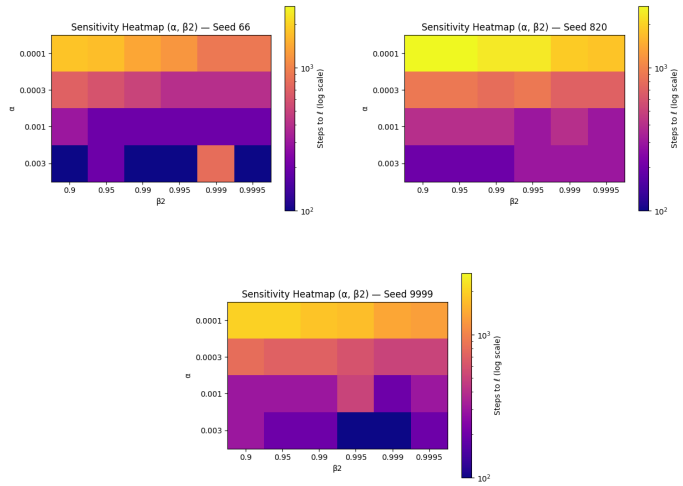


Figure 7: β_2-sweep Sensitivity Heatmap

second moment (the squared gradient). This grants the ability to tune the learning rates for the optimizer for each parameter during training. In this experiment, several Adam variants were evaluated to understand their contribution of its individual components: removing bias correction, setting $\beta_1 = 0$ to simulate an RMSProp kind of update, and using AdamW, which decouples weight decay from the gradient step. This comparison allows us to see how different components of adaptive optimization can influence convergence speed and generalization.

Sensitivity heatmaps were generated to analyze the effect of the Adam hyperparameters. They were gauged on the number of steps required to reach the validation-loss threshold (ℓ) for coarse grids over (α, β_1) and (α, β_2). Across all three seeds, the learning rate (α) had the strongest influence on convergence. Larger learning rates ($\alpha \in [0.001, 0.003]$) consistently achieved the fastest convergence, reducing the loss threshold within about 10^2 updates. Smaller learning rates ($\alpha = 10^{-4}$) required an order of magnitude more updates, signaling significantly slower optimization.

Both momentum parameters β_1 and β_2 had a more limited effect on optimization speed. Across most learning-rate settings, varying β_1 from 0 to 0.9 produced modest variations in the number of steps required for convergence. Sweeping β_2 from 0.9 to 0.9995 resulted in relatively small changes in performance as well, although some sensitivity appears for larger learning rates. This is indicative of Adam being relatively robust to their variation. This further suggests that practical tuning efforts should prioritize selecting an appropriate learning rate, while β_1 and β_2 can remain near standard defaults without significant performance degradation.

An important note is that these patterns were consistent across all three random seeds, indicating that the optimizer's sensitivity to the learning rate is robust to initialization. Overall, the heatmaps help portray how Adam performs reliably across a range of momentum settings, but remains dependent on the learning rate for achieving fast convergence.

4.2.2 Adam Variant Comparison

The Adam variants' performance was also evaluated to understand how their individual settings can influence training behavior. Figure 8 depicts the number of steps required to reach the fixed validation-loss threshold (ℓ) across all three seeds. Among the Adam methods, the variant without bias correction consistently reached the threshold in the fewest updates, often requiring $100 - 200$ steps. Standard Adam and AdamW

required slightly more updates (around $200 - 300$) to converge. This indicates that removing bias correction may lead to noisier parameter updates early in training due to biased moment estimates. This enables the optimizer to reach the validation-loss threshold more quickly, but often results in reduced stability later in training. In general, the Adam-based optimizers consistently reached the validation-loss threshold (ℓ) in fewer updates than SGD-based methods. Although the exact number of steps varied due to stochasticity in minibatch sampling and initialization, the overall optimization trends remained stable across all three seeds.

The validation-loss curves suggest that faster convergence may come at the expense of reduced stability. Although all Adam variants achieved similar validation-loss values early in training, the variant without bias correction tended to exhibit larger upward ticks later in training compared to the standard Adam and AdamW. In contrast, AdamW and Adam both maintained stable validation-loss trajectories across all three seeds, but both exhibited gradual increases in validation loss later in training.

The validation F1 curves also show the Adam variants achieving similar final performance levels, with the F1 scores stabilizing around 0.54. Even though optimizer design influences convergence speed, differences seen between variants was moderate which indicates limited impact on classification performance.

The generalization gap was measured as the difference between training and validation loss at the training budget. Across optimizers, the gap tended to increase later in training as validation loss began to increase inversely to training loss. Momentum-based SGD variants exhibited larger gaps than vanilla SGD, whereas standard Adam and AdamW maintained slightly smaller generalization gaps. This is in comparison to the variant without bias correction, which often showed rapid validation degradation after early convergence.

Adaptive optimizers like Adam reached the validation-loss threshold substantially faster than SGD-based methods, while maintaining similar final validation performance. Hyperparameter sensitivity analysis showed the significance of the learning rate (α) in Adam's convergence behavior, whereas the momentum parameters β_1 and β_1 mainly affected stability instead of convergence speed.

5 Regularization Study

In this experiment, the effects of regularization on neural network generalization was evaluated under a fixed optimizer and fixed compute. All experiments were conducted on the Adult dataset using the standard Adam optimizer with the best hyperparameters identified in the Adam Ablations (part 2). The network architecture and compute budget were held constant so any performance differences that arose would be attributed to regularization effects rather than optimization changes. Several regularization techniques were explored, including L2 weight decay, early stopping, dropout, and noise-based regularization methods (like label smoothing and input noise augmentation). For each regularization method, hyperparameter sweeps were conducted to determine the most effective regularizer (seen in *part3_sweeps.csv*), and then combined

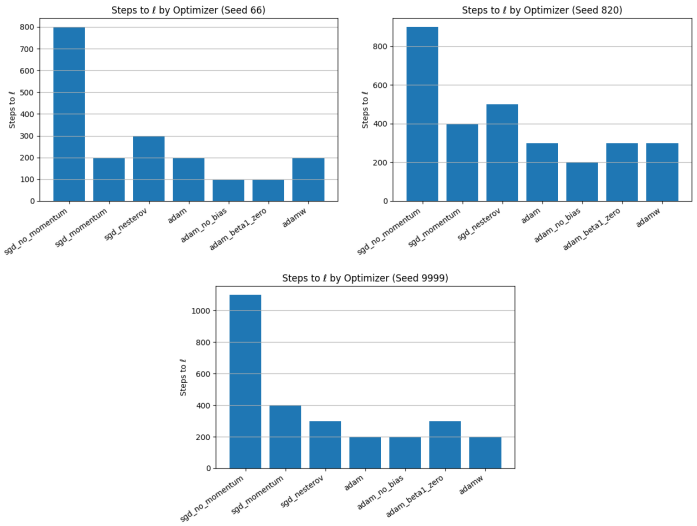


Figure 8: Steps to Validation-Loss Threshold (ℓ)

to determine whether combinations of regularization methods could improve generalization relative to the baseline (seen in *part3_summary.csv*).

5.1 Baseline Adam

The baseline model used the Adam optimizer without additional regularization and achieved a mean test F1 score of approximately 0.678 across all three seeds, as well as a mean test accuracy of 0.8013 and a mean test loss of 0.5741.

5.2 Best Single Regularizer

Each regularization technique was evaluated individually. Out of all of them, **label smoothing** with a strength 0.15 produced the best improvement, achieving a mean test F1 score of approximately 0.685 (across all three seeds). This represents an absolute improvement of roughly +0.007 over the baseline. Weight decay also improved performance slightly, whereas dropout, early stopping, and input noise augmentation only produced marginal gains.

5.3 Best Regularizer Combination

Combinations of regularization methods were evaluated under the same compute budget. The best regularizers were: weight decay = 0.001, input noise = 0.02, early stopping = 5, dropout = 0, and label smoothing = 0. The combination achieved a mean test F1 of 0.6764 ± 0.0041 across all three seeds, with a mean test loss of 0.5719. This setup indicates that moderate parameter shrinkage and mild input perturbation were sufficient to improve generalization, while adding more regularization mechanisms limited performance under the fixed compute budget.

An important discovery is that even though several regularizers individually improved performance, combining them did not help. The best combination in comparison to the baseline resulted in a lower mean test F1: 0.6764 (best) -0.6779 (baseline) $= -0.0015$.

The degradation observed in the combined regularization setup suggests over-regularization, where multiple constraints

simultaneously limit model capacity. While individual methods such as *label smoothing* improves generalization by reducing overconfidence, combining them with *weight decay* and *input noise* introduces competing regularization effects. Under the fixed compute budget, this prevents the model from sufficiently fitting the training data, leading to the observed reduced test performance. Compared to the optimizer improvements observed in Part 2, regularization provides smaller but more targeted gains, indicating that optimization primarily affects convergence while regularization fine-tunes generalization.

Table 3: Regularization sweep results on the Adult dataset using Adam with fixed architecture and compute budget

Method	Setting	Best Val Loss	Test F1
Adam (Baseline)	None	0.5560	0.6779
Weight Decay	$\lambda = 10^{-3}$	0.5531	0.6835
Dropout	$p = 0.3$	0.5593	0.6805
Label Smoothing	$\epsilon = 0.15$	**0.6129**	**0.6846**
Input Noise	$\sigma = 0.01$	0.5550	0.6793
Early Stopping	patience = 5	0.5558	0.6746
Best Combination	wd=0.001, noise=0.02, ES=5	0.5535	0.6764

Table 4: Summary of baseline and best regularized Adam models (Adult dataset)

Method	Best Val Loss	Test F1	Updates	Notes
Adam Baseline	0.5560	0.6779	567	No regularization
Best Single Regularizer	0.6128	0.6845	1167	Label Smoothing ($\epsilon = 0.15$)
Best Combination	**0.5535**	**0.6764**	**550**	wd + input noise + ES

6 Conclusion

This report examined how different optimization strategies and regularization techniques influenced the training dynamics and performance of neural networks on the Adult Income and Wine Quality datasets. Randomized Optimization methods (Random Hill Climbing, Simulated Annealing, and Genetic Algorithms) were able to improve validation loss in some cases when applied to the final layers of the network, but proved to be generally less competitive overall compared to gradient-based training. This was due to their limited search efficiency in high-dimensional parameter spaces. Under matched compute budgets, these methods achieved lower test performance (F1 ≈ 0.55) compared to gradient-based methods (F1 ≈ 0.68), reflecting their limited search efficiency in higher-dimensional parameter spaces.

The optimizer ablation experiments on the Adult dataset demonstrated that adaptive gradient methods (including Adam and its variants), consistently outperformed SGD-based methods. The Adam methods achieved faster convergence to the validation-loss threshold (ℓ) and more stable training across multiple seeds. The analysis further showed that actually removing key components of an optimizer (like momentum or bias correction), often degraded the early training stability and performance.

The regularization study revealed that explicit regularization techniques had an impact on network generalization. Techniques like L2 weight decay and input noise improved validation and test metrics in comparison to the baseline Adam optimizer. This is indicative of the importance of controlling the model's capacity and overfitting rather than solely choosing the best optimizer. Combining multiple regularization methods resulted in a slight degradation in performance, suggesting over-regularization under a fixed compute budget. This highlights that while optimization governs convergence behavior, regularization plays a more targeted role in improving generalization.

The experimental results partially support the stated hypotheses. For the Adult dataset, adaptive optimizers such as Adam achieved faster convergence and more stable performance than SGD, confirming the expected advantage of momentum and adaptive updates. Improvements in test F1 score were likely driven by regularization choices rather than optimizer selection alone. On the contrary, randomized optimization methods did not outperform the gradient-based baseline, supporting the hypothesis that their benefits are limited to small-scale parameter tuning. For the Wine dataset, increased variability across seeds was observed, consistent with the hypothesis that smaller datasets amplify stochastic effects in optimization. However, contrary to the hypothesis, randomized optimization methods did not outperform the gradient-based baseline on test performance.

Overall, these experiments suggested a practical training strategy for building machine learning systems. By first achieving stable convergence with a model and a well-configured adaptive optimizer (like Adam), one can apply regularization techniques to improve model generalization. Randomized optimization may also provide useful exploratory behavior in determining the best weight settings, but gradient-based optimization remains the most effective approach to training neural networks of this scale.

7 References

- Tam, Adrian. "Building Multilayer Perceptron Models in PyTorch - MachineLearningMastery.com." MachineLearningMastery.com, 26 Jan. 2023, machinelearningmastery.com/building-multilayer-perceptron-models-in-pytorch/.
- Zhang, Xinhe. "Multi-Layer Perceptron (MLP) in PyTorch." Deep Learning Study Notes, 26 Dec. 2019, medium.com/deep-learning-study-notes/multi-layer-perceptron-mlp-in-pytorch-21ea46d50e62.
- John Mansfield. "Pyperch/Pyperch/Optim at Master · Jlm429/Pyperch." GitHub, 2025, github.com/jlm429/pyperch/tree/master/pyperch/optim.
- GeeksforGeeks. "Hill Climbing in Artificial Intelligence." GeeksforGeeks, 12 Dec. 2017, www.geeksforgeeks.org/artificial-intelligence/introduction-hill-climbing-artificial-intelligence/.
- Wikipedia Contributors. "Simulated Annealing." Wikipedia, Wikimedia Foundation, 4 Nov. 2019, en.wikipedia.org/wiki/Simulated_annealing.
- Brownlee, Jason. "Simulated Annealing from Scratch in Python." Machine Learning Mastery, 18 Feb. 2021, machinelearningmastery.com/simulated-annealing-from-scratch-in-python/.
- GeeksforGeeks. "Genetic Algorithms." GeeksforGeeks, 29 June 2017, www.geeksforgeeks.org/dsa/genetic-algorithms/.
- Brownlee, Jason. "Simple Genetic Algorithm from Scratch in Python." Machine Learning Mastery, 2 Mar. 2021, machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/.
- GeeksforGeeks. "What Is Adam Optimizer?" GeeksforGeeks, 22 Oct. 2020, www.geeksforgeeks.org/deep-learning/adam-optimizer/.
- Kingma, Diederik P, and Jimmy Ba. "Adam: A Method for Stochastic Optimization." ArXiv, 22 Dec. 2014, arxiv.org/abs/1412.6980.
- Brownlee, Jason. "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning - MachineLearningMastery.com." MachineLearningMastery.com, 2 July 2017, www.machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.
- "Why Is It Important to Include a Bias Correction Term for the Adam Optimizer for Deep Learning?" Cross Validated, 2016, stats.stackexchange.com/questions/232741/why-is-it-important-to-include-a-bias-correction-term-for-the-adam-optimizer-for.
- "Deep Learning: How Does Beta_1 and Beta_2 in the Adam Optimizer Affect It's Learning?" Cross Validated, 4 Mar. 2017, stats.stackexchange.com/questions/265400/deep-learning-how-does-beta-1-and-beta-2-in-the-adam-optimizer-affect-its-lear.
- GeeksforGeeks. "Regularization in Machine Learning." GeeksforGeeks, 23 May 2019, www.geeksforgeeks.org/machine-learning/regularization-in-machine-learning/.
- C, Bala Priya. "Regularization in Neural Networks — Pinecone." Www.pinecone.io, 30 June 2023, www.pinecone.io/learn/regularization-in-neural-networks/.
- Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." Journal of Machine Learning Research, vol. 15, 2014, pp. 1929–1958, www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_content=buffer79b4.
- GeeksforGeeks. "F1 Score in Machine Learning." GeeksforGeeks, 27 Dec. 2023, www.geeksforgeeks.org/machine-learning/f1-score-in-machine-learning/.
- PyTorch Documentation (https://docs.pytorch.org/docs/stable/index.html)