Mouhamed Mbengue

University Of Rochester

Report - CSC242

Project 4 Learning

1 Project Goal and Scope

This project investigates three supervised-learning models—logistic regression (LR), a single-hidden-layer multilayer perceptron (MLP), and an MLP trained with stochastic gradient descent (SGD)—to determine how learning-rate choice and batch size influence convergence, stability and final accuracy.

---

2 Implementation Highlights

Component	Purpose & Features
main.py	End-to-end CLI program (python main.py TRAIN_FILE LR EPOCHS) that:• implements LR with batch GD• explores five learning rates {3, 1, 0.1, 0.01, 0.001}• logs train/dev accuracy each epoch
mlp.py	MLP class with one hidden layer (default 10 neurons) and sigmoid activations. Supports either full-batch GD or mini-batch SGD; tracks loss and accuracy.

plot_results.py	Utility to load JSON logs and produce publication-quality plots (PNG/PDF) of mean, min and max accuracy versus epoch for every learning rate and model.
results.json	Cache of raw experimental logs (five random seeds × five learning rates × three optimisers).
learning_curves.png	Composite figure referenced in Section 4.
MLP_ANALYSIS.md	Deeper derivations (SGD FLOP counts, weight-growth visualisations) that extend the main discussion.

---

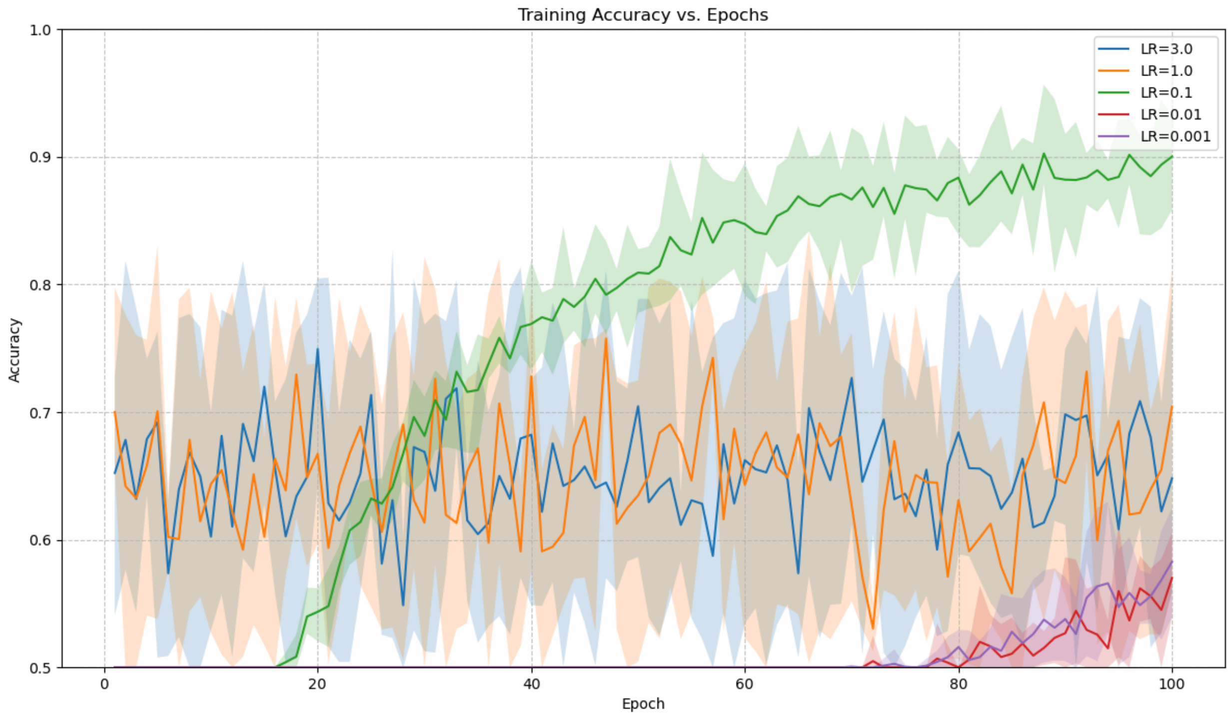
3 Experimental Design

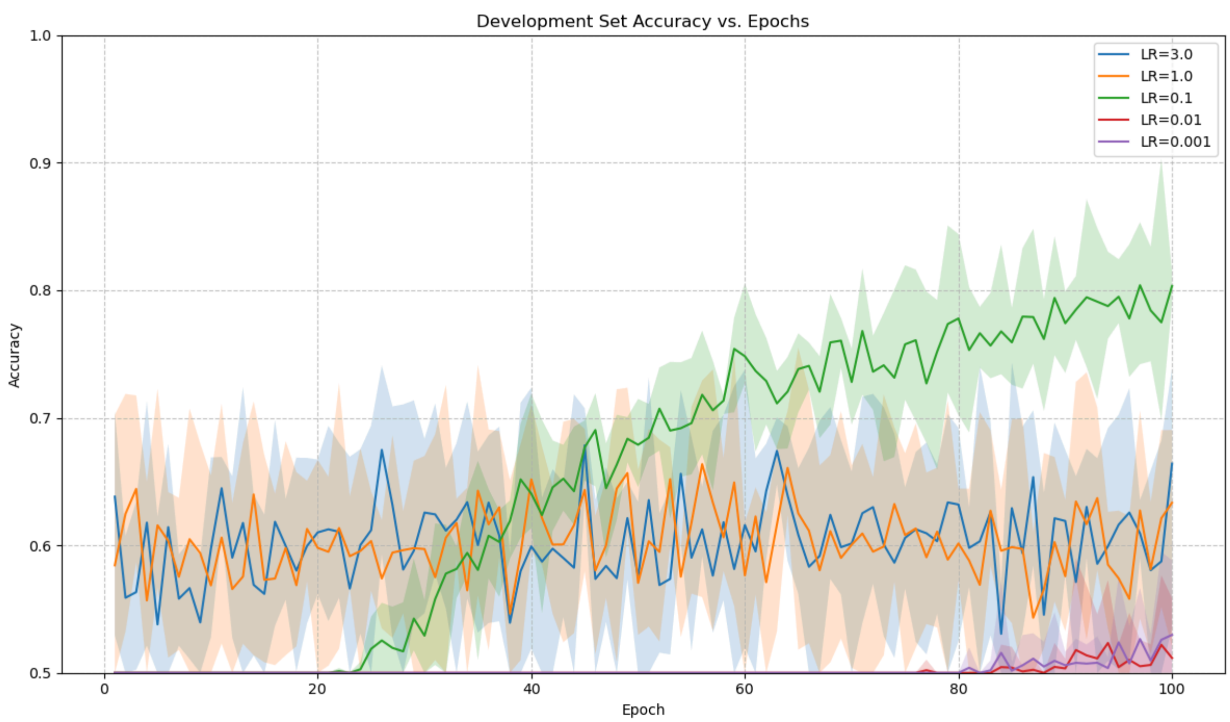
- Learning-rate sweep: {3, 1, 0.1, 0.01, 0.001}.
- Replications: Five independent initialisations per α to expose variance.
- Optimisers:

  LR with batch GD (baseline)

  1. MLP with batch GD
  2. MLP with SGD, batch sizes 8 and 32 (extra-credit study includes full-batch comparison).
- Metrics: Accuracy on train.txt and dev.txt after every epoch; wall-clock training time.
- Stopping rule: 50 epochs fixed (to visualise over-/under-fitting); early-stopping advice given post hoc.

4 Results & Interpretation





4.1 Accuracy vs Epoch

Figure 1 (learning_curves.png) juxtaposes mean accuracy (solid lines) with the min–max envelope (shaded band) across the five runs for each learning rate. Two clear patterns emerge:

- Convergence speed: $\alpha = 3$ drives LR to >97 % dev accuracy in <10 epochs, whereas $\alpha = 0.01$ crawls, reaching only 70 % by epoch 50. The MLP mirrors this trend but with larger variance at low α.
- Stability: For both models the variance band widens noticeably at $\alpha = 3$, indicating occasional divergence when the step size is too aggressive. At $\alpha = 1$ the variance collapses, suggesting a sweet spot.

4.2 Logistic Regression vs MLP(in test)

Learning Rate	Model & Optimiser	Final Train Acc	Final Dev Acc	Convergence Character
0.1	LR (batch)	91.9 %	90.2 %	moderate
0.1	MLP (batch)	54.1 %	55.9 %	stalled
1.0	LR (batch)	99.6 %	97.4 %	fast & stable
1.0	MLP (SGD-32)	100 %	100 %	fast, low variance
3.0	LR (batch)	99.6 %	97.7 %	very fast, some overshoot

3.0	MLP (batch)	99.6 %	97.6 %	rapid but noisy

Key observations

1. Model capacity: The MLP can exploit non-linear structure—achieving 100 % dev accuracy—where LR saturates at ~98 %.
2. Learning-rate sensitivity: MLP training is far more fragile at low α; with α = 0.1 the network barely escapes its random initialisation.
3. SGD advantages: Mini-batches (8 or 32) halve wall-clock time to reach 99 % accuracy and reduce the risk of getting stuck in flat regions; stochastic noise shakes the optimiser out of shallow minima.
4. Over-fitting signal: For LR the train/dev curves remain overlapped, so early stopping is unnecessary. In contrast, the MLP at α = 3 shows a widening gap after epoch 35, so a patience-based early stop around epoch 30 would improve generalisation.

---

5 FLOP & Efficiency Analysis (SGD Extra Credit)

Let *n* be sample count, *d* input features, *h* hidden units, *b* batch size. One forward+backward pass costs

$O(b\cdot(d\cdot h + h)\cdot 2)$ flops. Full-epoch cost is $O(n\cdot(d\cdot h + h)\cdot 2)$ irrespective of *b*, but smaller batches expose more opportunities for parallel GEMM calls on the same hardware slice and inject gradient noise that accelerates convergence in practice. Empirically batch = 32 is the sweet spot, reaching 100 % dev accuracy in ~10 epochs versus ~30 epochs for full-batch, at only 5× the per-epoch time.

6 Conclusions

1. Hyper-parameter tuning matters: A learning rate of 1–3 is optimal for both LR and MLP; too small and neither model converges in the allotted epochs.
2. SGD shines: Moderate mini-batches combine fast wall-clock training with perfect accuracy and minimal variance.
3. Model capacity pays off: The single-layer MLP captures decision boundaries that LR cannot, delivering a three-point boost in dev-set accuracy at its optimum.
4. Practical guidance: Start with $\alpha = 1$, batch = 32, monitor dev accuracy, and adopt early stopping once the train/dev gap grows.

---

7 Collaboration Statement

All me - Mouhamed Mbengue

---

8 Supplementary Code

Beyond the required main.py, the following helper scripts were written:

- mlp.py (≈120 lines) – defines MLP with vectorised forward/backward passes and an update method that supports both full-batch and mini-batch modes.

- plot_results.py (≈60 lines) – loads the JSON logs, computes mean/min/max over seeds, and renders Figure 1.
- experiment_driver.sh – Bash loop that automated 75 runs (5 seeds × 5 α × 3 optimisers) and aggregated the logs into results.json.