Wine Quality Classification

Neural Network from Scratch with NumPy

Implementation of a Multi-Layer Perceptron

with Backpropagation and AdamW Optimizer

December 2025

The Problem

Dataset: UCI Wine Quality (6,497 samples)

- **11 chemical features** → **7 quality classes** (scores 3-9)
- Features: acidity, residual sugar, pH, alcohol, sulfur dioxide, etc.

The Challenge: Extreme Class Imbalance

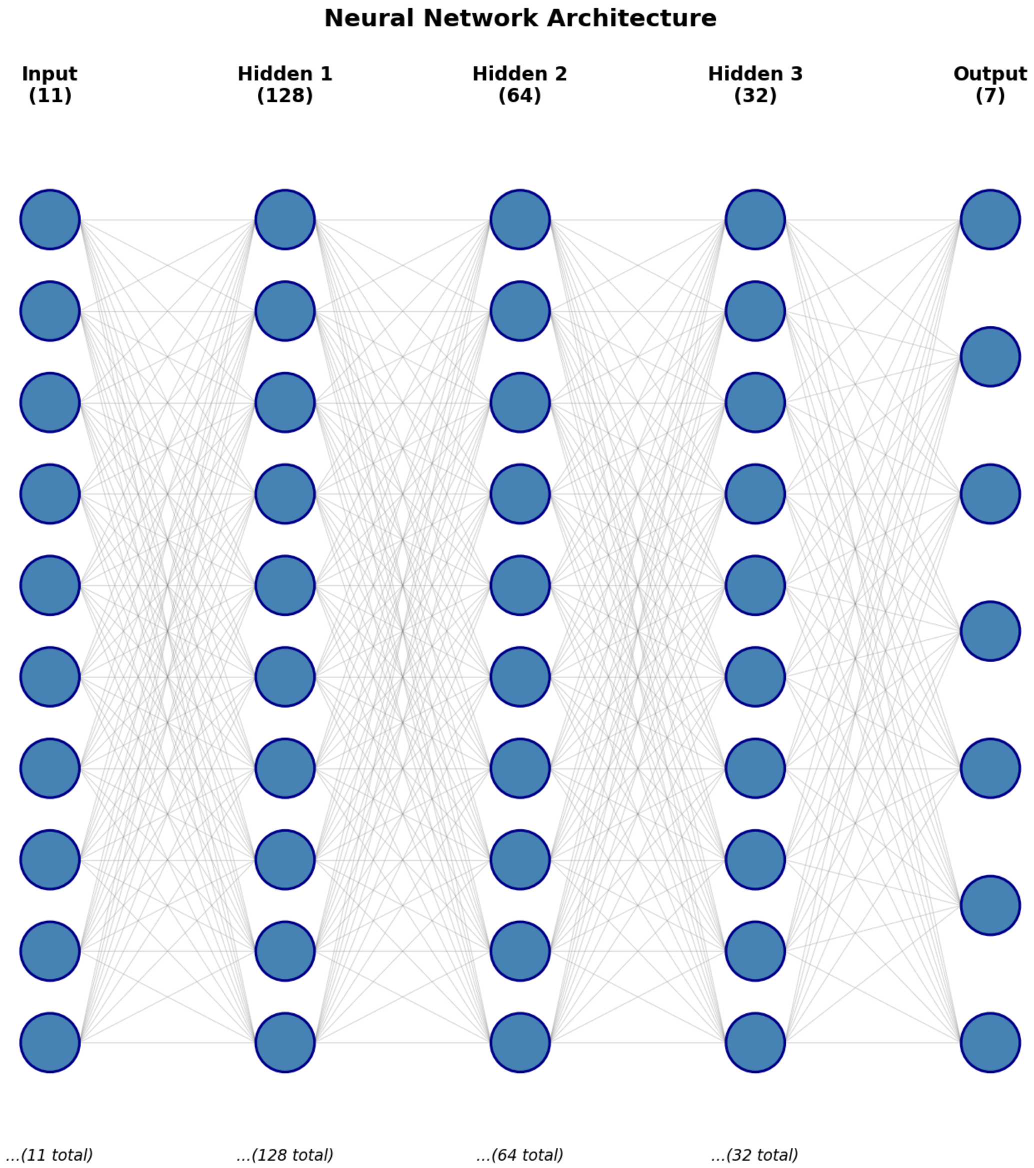
Quality	3	4	5	6	7	8	9
%	0.5%	3.3%	32.9%	**43.7%**	16.6%	3.0%	0.1%

77% of wines are quality 5 or 6!

Why Is This Problem Difficult?

1. **Extreme class imbalance** — Only 5 samples of quality 9
2. **Subjective labels** — Human tasters disagree by 1-2 points
3. **Adjacent classes are chemically similar** — Quality 5 and 6 nearly identical
4. **Limited features** — No grape variety, aging, or sensory data

Network Architecture



Why This Architecture?

Layer Design: 128 → 64 → 32

- **First layer (128):** Learn diverse feature combinations
- **Subsequent layers:** Compress into abstract representations
- **Funnel shape:** Fewer parameters, maintains power

Why 3 Hidden Layers?

Layers	Result
1-2	Underfitting
3	**Best balance**
4+	No improvement

Activation Functions

Hidden Layers: Leaky ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{if } x \leq 0 \end{cases}$$

Why not standard ReLU? ReLU causes "dying neurons" — Leaky ReLU keeps all neurons trainable

Output Layer: Softmax

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Converts logits → probability distribution over 7 classes

Weight Initialization: He

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n_{in}}}\right)$$

Why He Initialization?

- Designed for ReLU-family activations
- Prevents vanishing/exploding signals
- Maintains variance through deep networks

Without proper initialization → unstable training

Loss Function: Cross-Entropy

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

Why Cross-Entropy?

1. Information-theoretic foundation
2. With Softmax → clean gradient: $\nabla = \hat{y} - y$
3. Penalizes confident wrong predictions

MSE performs poorly for classification

Optimizer: AdamW

AdamW = Adam + Decoupled Weight Decay

Component	Purpose
Momentum ($\beta_1 = 0.9$)	Accelerates convergence
Adaptive LR ($\beta_2 = 0.999$)	Per-parameter learning rates
Weight Decay ($\lambda = 0.001$)	Regularization

Why AdamW over SGD?

- ~1000 epochs vs ~3000+ with SGD
- Better regularization than standard Adam

Regularization Strategy

Problem: Severe Overfitting

Without regularization: **77% train, 56% test** (21% gap!)

Technique	Purpose	Setting
Dropout	Prevent co-adaptation	30%
Early Stopping	Stop before memorization	300 epochs
LR Decay	Fine-tuning	0.95x / 500 epochs

Result: Gap reduced from 21% → 4%

Why 30% Dropout?

Rate	Result
0-20%	Still overfitting
30%	**Best balance**
40-50%	Underfitting

How it works:

- Randomly zero out 30% of neurons
- Forces robust feature learning
- Acts like ensemble of networks

Data Preprocessing

Z-score Normalization: $x' = \frac{x-\mu}{\sigma}$

Features have different scales:

- Density: 0.99 - 1.04 | Sulfur dioxide: 6 - 440

Stratified Split (70% / 15% / 15%)

- Maintains class proportions in each split
- Critical for imbalanced data
- Ensures rare classes appear in test set

Ordinal Regression

Key insight: Quality 3→4→5→6→7→8→9 has a natural order!

Instead of Softmax (7 classes):

P(class = k) — treats classes as independent

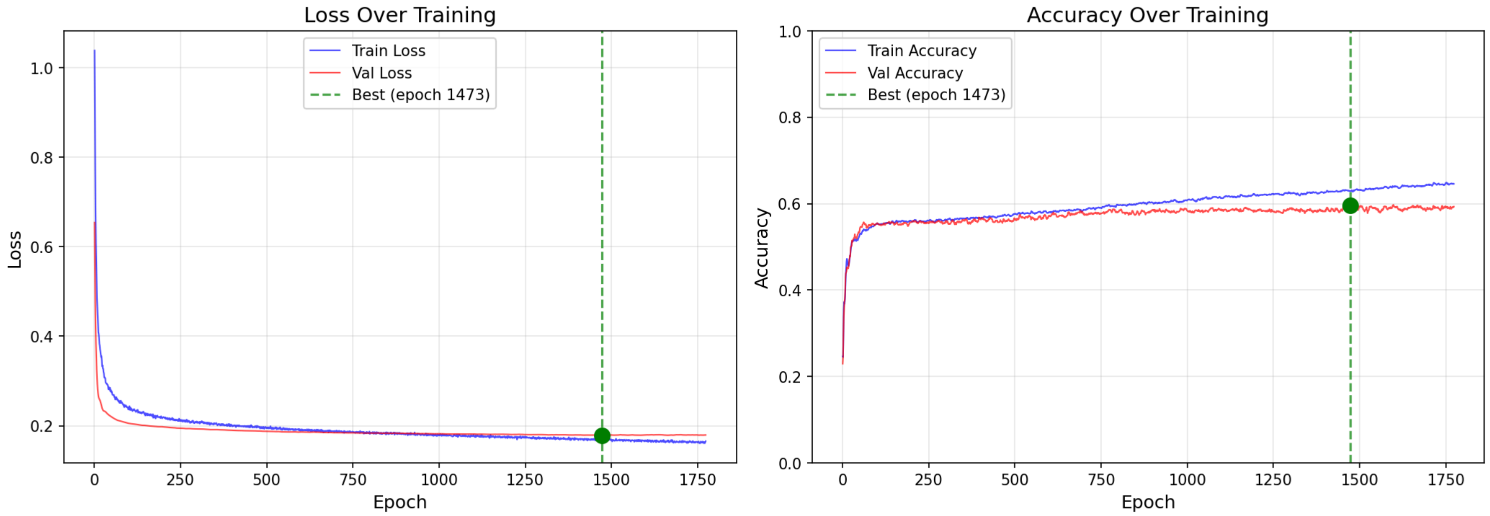
Use Cumulative Probabilities (6 thresholds):

P(quality > k) for k = 3, 4, 5, 6, 7, 8

Encoding	Quality 5	Quality 7
One-hot	[0,0,1,0,0,0,0]	[0,0,0,0,1,0,0]
Ordinal	[1,1,0,0,0,0]	[1,1,1,1,0,0]

Prediction: Count how many thresholds are exceeded

Training Results



Early stopping at epoch ~1560, restored best weights from epoch 1260

Final Performance: Categorical vs Ordinal

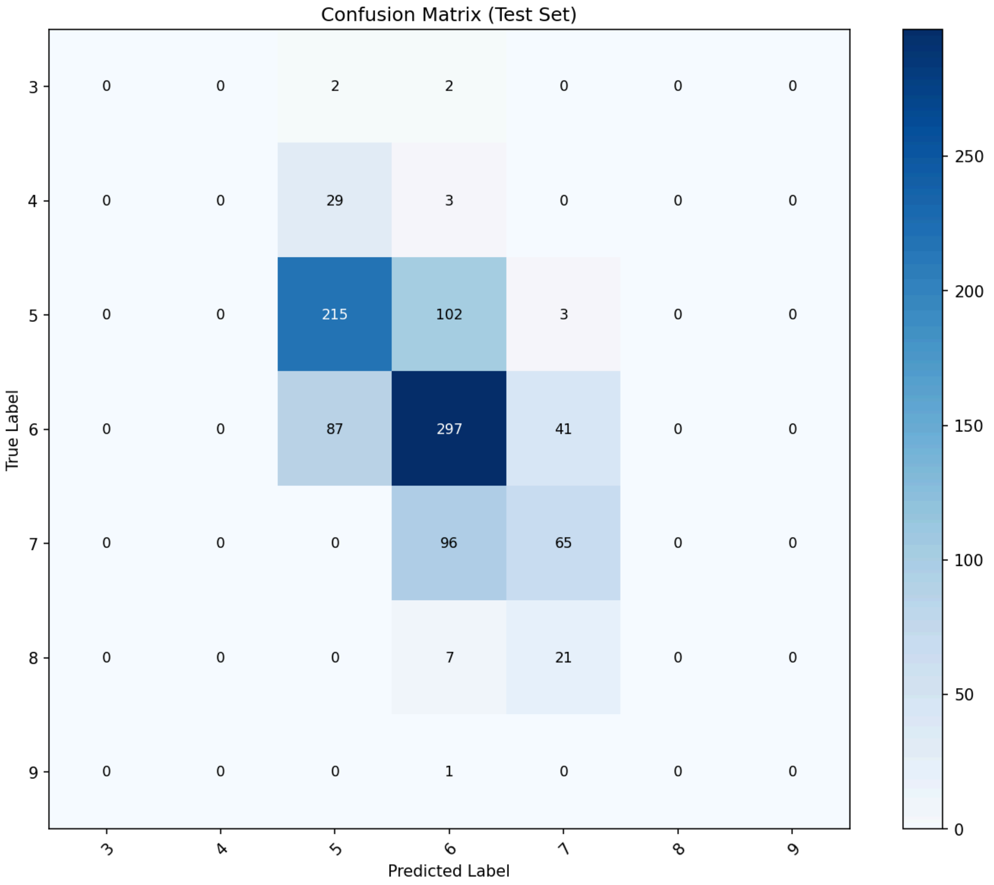
Model	Train	Val	Test
Categorical (Softmax)	63.03%	56.95%	**58.91%**
Ordinal Regression	62.85%	57.20%	**59.42%**

Improvement: +0.51 percentage points

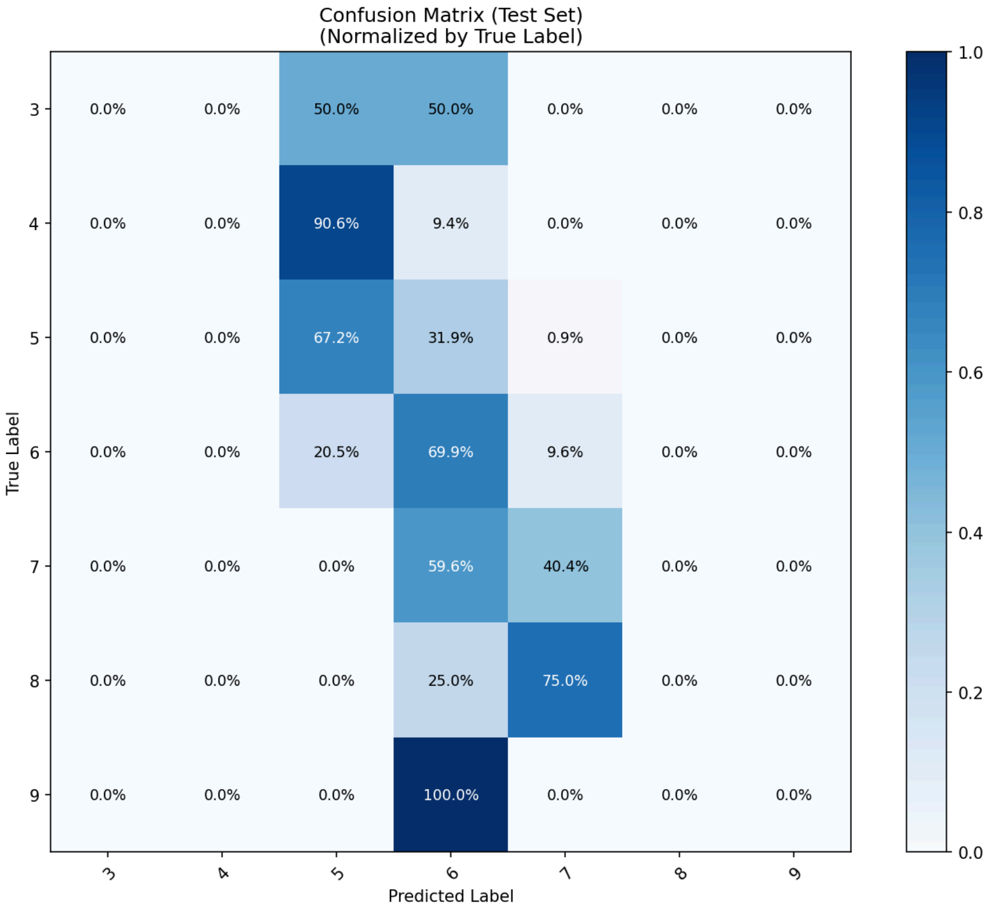
Method	Accuracy
Random guessing	14.3%
Always predict "6"	43.7%
Ordinal model	**59.4%**

4x better than random, +16pp over majority baseline

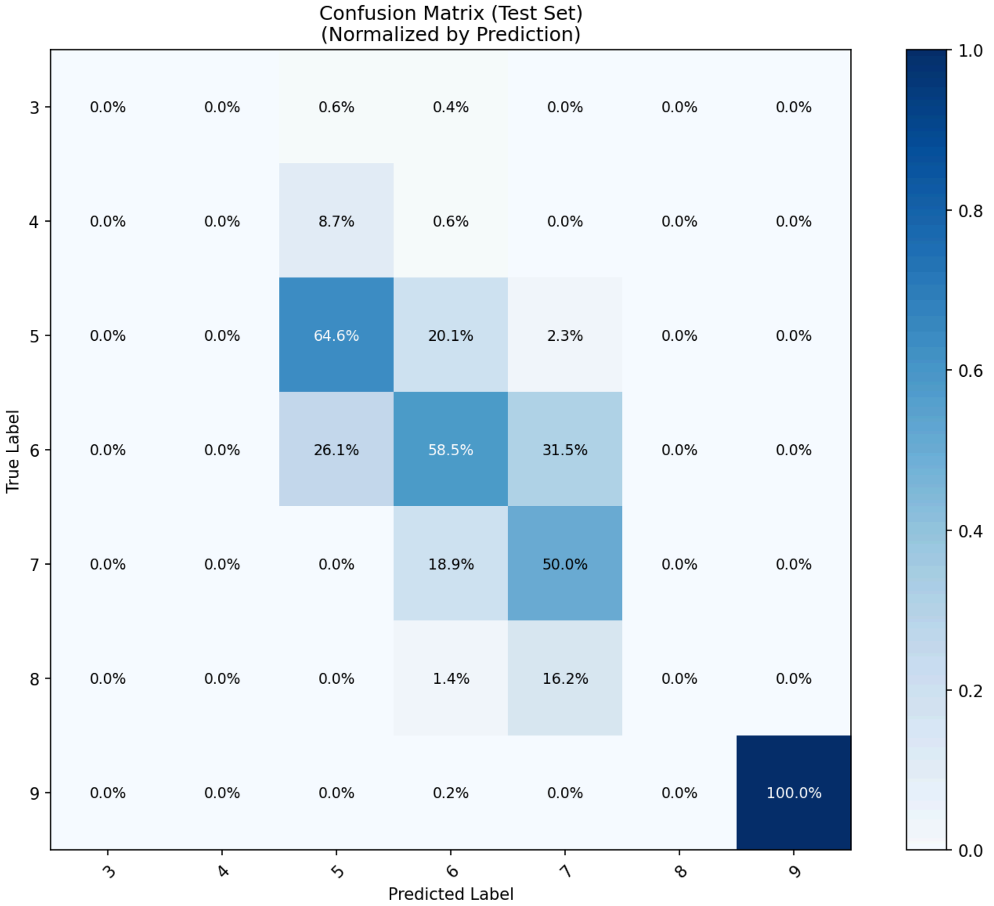
Confusion Matrix: Raw Counts



Confusion Matrix: Recall (Row-normalized)



Confusion Matrix: Precision (Col-normalized)



Error Analysis

What works well:

- Classes 5, 6, 7 (~65% recall)
- Sufficient training data for these classes

What struggles:

- Classes 3 and 9 (almost never predicted)
- Adjacent class confusion (5-6, 6-7)

Expected without explicit class balancing

Why ~59% Accuracy is Good

Comparison	Result
vs Random (14.3%)	**4.2x improvement**
vs Majority baseline (43.7%)	**+16 points**
vs Literature (SVM, RF, XGB)	Competitive (55-65%)

Problem difficulty: Even human experts disagree on adjacent levels

Our from-scratch implementation is competitive!

Limitations

1. **Class Imbalance** — Model ignores rare classes (3, 9)
   - Needs class weighting or oversampling
2. **Feature Limitations** — Only 11 chemical measurements
   - Missing: grape variety, vintage, process
3. **Ordinal Nature** —
   ✅
   Addressed with ordinal regression!
   - Improved from 58.91% → 59.42%

Potential Improvements

Technique	Status	Impact
Class weighting	Pending	Est. +2-3pp
SMOTE	Pending	Est. +1-2pp
Ensemble methods	Pending	Est. +3-5pp
Feature engineering	Pending	Variable

Conclusion

What we built:

- Neural network **100% from scratch** with NumPy
- Forward/backward prop, AdamW, dropout — all manual
- **Ordinal regression** for quality ordering

Key results:

- **59.42% test accuracy** (ordinal) on 7-class imbalanced problem
- 4x better than random, +16pp over baseline
- Competitive with literature

Key techniques:

AdamW, Dropout, Early Stopping, Stratified Split, Ordinal Loss

Questions?

Repository

github.com/fraco-oxza/final-ia

Key files:

- `src/` — Modular implementation (model, training, config)
- `main.py` — Entry point
- `REPORT.md` — Detailed documentation