# 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

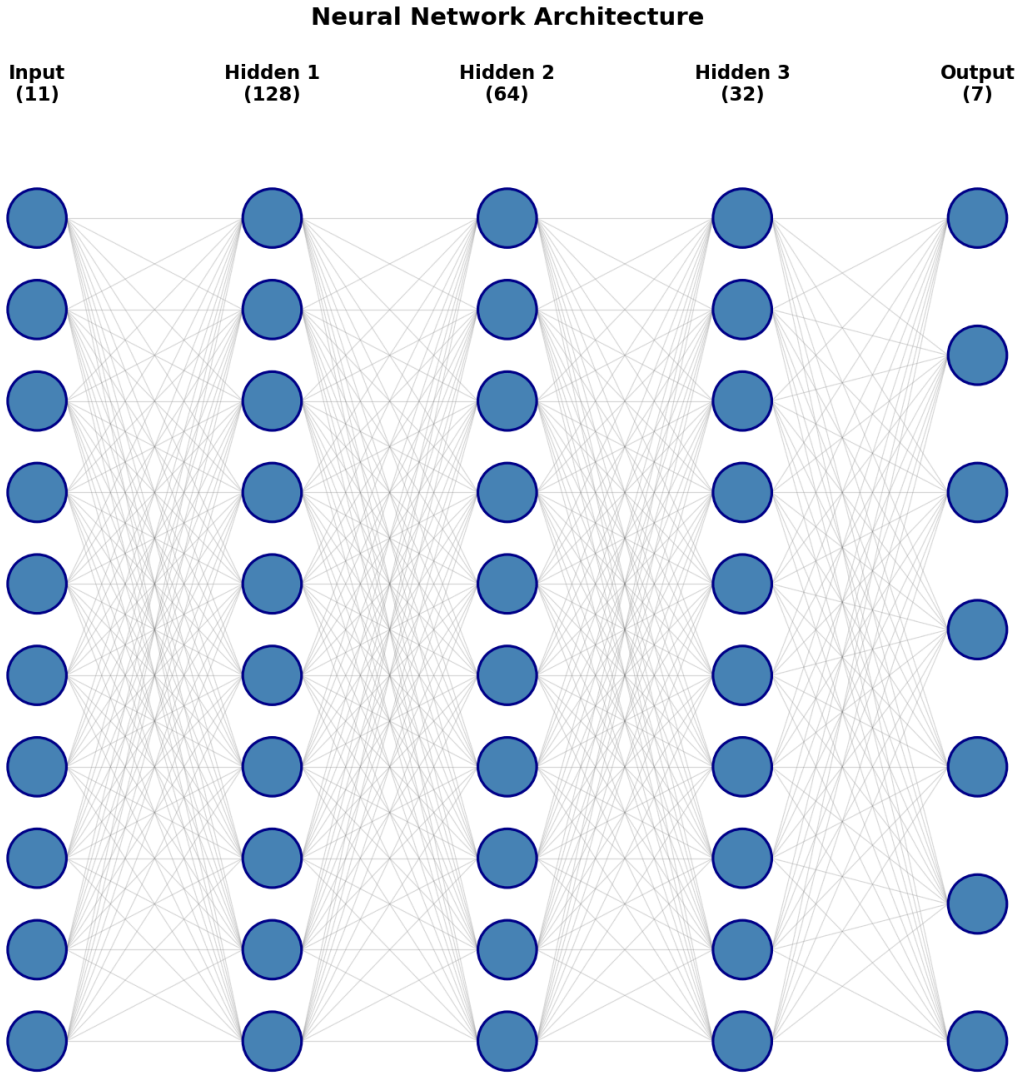
| Quality | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|
| % | 0.5% | 3.3% | 32.9% | **43.7%** | 16.6% | 3.0% | 0.1% |

**77% of wines are quality 5 or 6!**

# Why Is This Problem Difficult?

1. **Extreme class imbalance**
    - Only 5 samples of quality 9 in entire dataset
    - Model tends to ignore rare classes
2. **Subjective labels**
    - Human tasters may disagree by 1-2 points
3. **Adjacent classes are chemically similar**
    - Quality 5 and 6 wines have nearly identical compositions
4. **Limited features**
    - No grape variety, aging process, or sensory data

# Network Architecture



# Why This Architecture?

## Layer Design: 128 → 64 → 32

- **First layer (128):** High capacity to learn feature combinations
- **Subsequent layers:** Compress into abstract representations
- **Funnel shape:** Reduces parameters while maintaining power

## Why 3 Hidden Layers?

- 1-2 layers: Underfitting
- 3 layers: Best balance ✓
- 4+ layers: No improvement, slower training

# 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 can cause "dying neurons" (gradient = 0 forever)
- Leaky ReLU ensures all neurons remain trainable

### Output Layer: Softmax

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

# Weight Initialization

## He Initialization

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

## Why He?

- 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. Combined with Softmax $\rightarrow$ clean gradient: $\nabla = \hat{y} - y$
3. Properly penalizes confident wrong predictions

MSE performs poorly for classification tasks

# 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?

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

# Regularization Strategy

## The 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 patience |
| **LR Decay** | Fine-tuning | 0.95x every 500 epochs |

**Result:** Gap reduced from 21% → 4%

# Why 30% Dropout?

Tested multiple dropout rates:

| Rate | Result |
|---|---|
| 0-20% | Still significant overfitting |
| **30%** | **Best balance** ✓ |
| 40-50% | Underfitting, lower accuracy |

**How it works:**

- Randomly zero out 30% of neurons during training
- Forces network to learn robust features
- Acts like training ensemble of networks

# Data Preprocessing

## Z-score Normalization

$$x' = \frac{x - \mu}{\sigma}$$

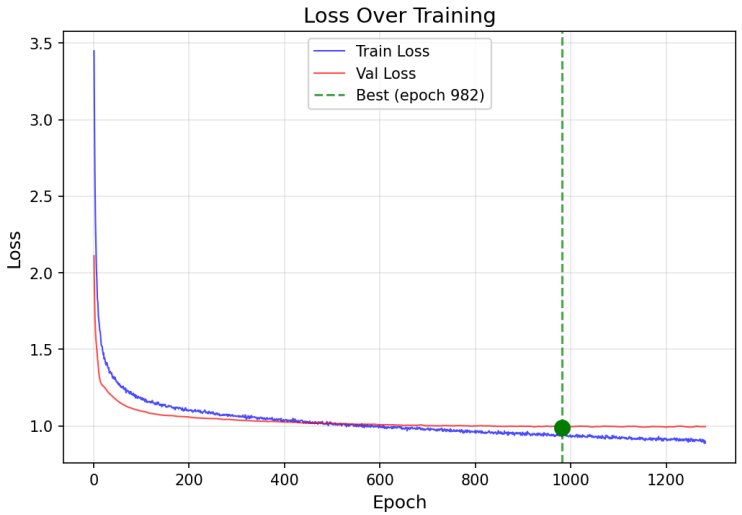
Features have different scales:

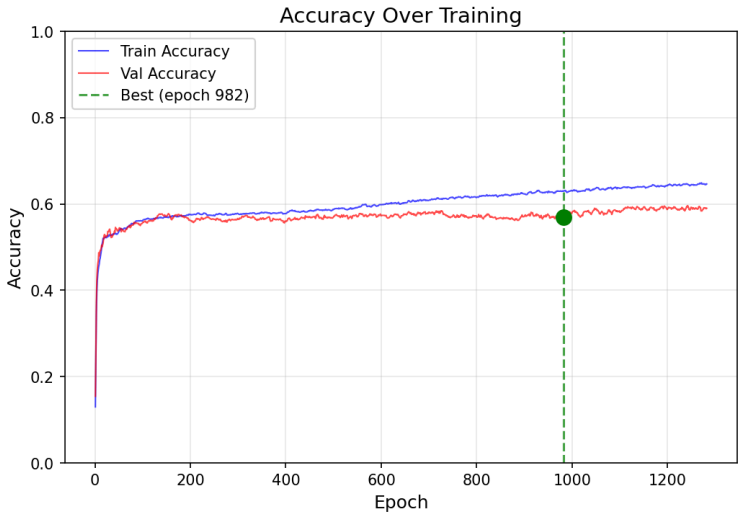
- Density: 0.99 - 1.04
- Sulfur dioxide: 6 - 440

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

- Maintains class proportions in each split

# Training Results




Early stopping at epoch ~1282, restored best weights from epoch 982

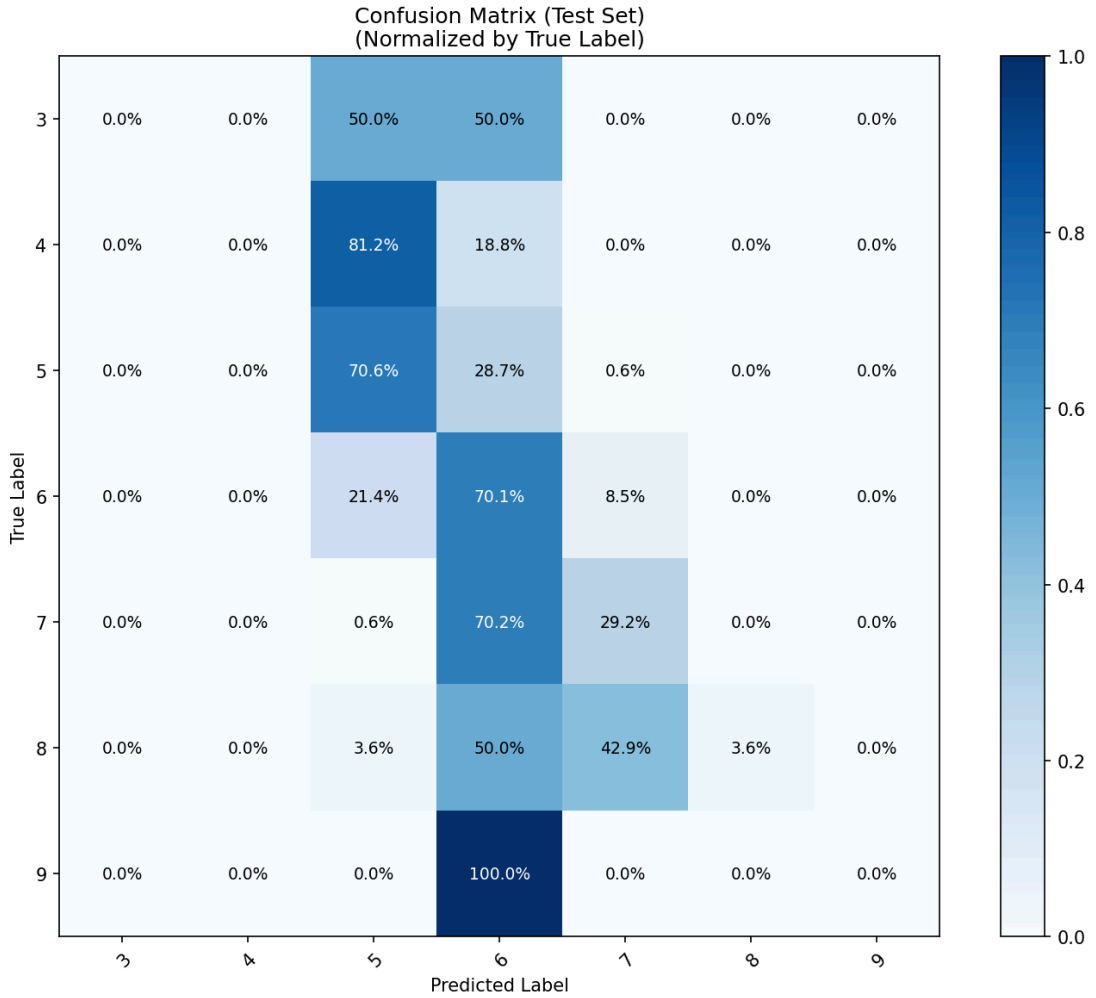
# Final Performance

| Metric | Value |
|---|---|
| Training Accuracy | 63.03% |
| Validation Accuracy | 56.95% |
| **Test Accuracy** | **58.91%** |

## Comparison with Baselines

| Method | Accuracy |
|---|---|
| Random guessing | 14.3% |
| Always predict "6" | 43.7% |
| **Our model** | **58.9%** |

# Confusion Matrix Analysis



# Error Analysis

**What the model does well:**

- Classes 5, 6, 7 (sufficient training data)
- ~65% recall on majority classes

**What the model struggles with:**

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

# Why ~59% Accuracy is Good

1. **vs Random:** 14.3% → 58.9% = **4.1x improvement**
2. **vs Majority baseline:** 43.7% → 58.9% = **+15 points**
3. **vs Literature:** Published results with SVM, Random Forest, XGBoost achieve 55-65%
4. **Problem difficulty:** Even human experts disagree on adjacent quality levels

**Our from-scratch implementation is competitive!**

# Limitations

1. **Class Imbalance**
    - Model ignores extremely rare classes (3, 9)
    - Would need class weighting or oversampling
2. **Feature Limitations**
    - Only 11 chemical measurements
    - Missing: grape variety, vintage, winemaking process
3. **Ordinal vs Categorical**
    - We treat quality as 7 independent classes
    - Being wrong by 1 should cost less than wrong by 5

# Potential Improvements

| Technique | How it helps |
|---|---|
| **Class weighting** | Penalize minority class errors more |
| **SMOTE** | Generate synthetic samples for rare classes |
| **Ordinal regression** | Respect the natural order of quality |
| **Ensemble methods** | Combine multiple models |
| **Feature engineering** | Polynomial features, interactions |

# Conclusion

**What we built:**

- Neural network **100% from scratch** with NumPy
- Forward/backward propagation, AdamW, dropout — all manual

**Key results:**

- **~59% test accuracy** on 7-class imbalanced problem
- Significantly outperforms baselines
- Competitive with literature results

# Questions?

**Repository**

github.com/fraco-oxza/final-ia

**Key files:**

- `neural_network.py` — Full implementation
- `REPORT.md` — Detailed documentation