Optimization Methods For Data Science

Final Project

Multi-Layer Perceptron for Age Regression

Milad Torabi
Student ID: 2103454

December 16, 2024

Contents

1 Problem Description

We aim to build a Multi-Layer Perceptron (MLP) to predict a target variable (age) from a set of input features. We minimize a **regularized L2 loss function** that combines the Mean Squared Error (MSE) with an L2 penalty on the weights. Because the hidden layers employ non-linear activation functions (ReLU, Tanh, Swish), the optimization landscape is *non-convex*.

Objective Function

$$E(\boldsymbol{\omega}, \mathbf{b}) \;=\; \frac{1}{N}\sum_{i=1}^{N}\big(y_i - \hat{y}_i\big)^2 \;+\; \lambda \sum_{l=1}^{L} \|\boldsymbol{\omega}^{(l)}\|_2^2.$$

- **Loss Term**:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

- **Regularization Term**:

$$\lambda \sum_{l=1}^{L} \|\boldsymbol{\omega}^{(l)}\|_2^2.$$

Here, $\hat{y}_i$ denotes the MLP's prediction for sample i, y_i is the true age, and $\lambda > 0$ is the regularization hyperparameter.

2 Data Preprocessing

The dataset used in this project is `AGE_REGRESSION.csv`. We divided it into feature columns $\mathbf{X}$ and the target variable `gt` (representing age). Z-score normalization was applied using training-set statistics to ensure consistency across both training and test sets, thereby mitigating issues caused by differing feature scales.

3 Optimization Goal and Strategy

We solve the following optimization problem:

$$\min_{\boldsymbol{\omega},\mathbf{b}} \quad E(\boldsymbol{\omega}, \mathbf{b}) \;=\; \min_{\boldsymbol{\omega},\mathbf{b}} \left[\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \;+\; \lambda \sum_{l=1}^{L} \|\omega^{(l)}\|_2^2\right].$$

Since the presence of non-linear activations makes the loss *non-convex*, standard gradient-based methods may encounter multiple local minima or saddle points. We employ **L-BFGS-B** (Limited-memory BFGS), as implemented by `scipy.optimize.minimize`. L-BFGS-B approximates the Hessian using limited memory, often converging faster than plain gradient descent in non-convex landscapes.

3.1 Loss and Gradient Computation

Loss:

$$E(\boldsymbol{\omega}, \mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2 + \lambda \sum_{l=1}^{L} \|\boldsymbol{\omega}^{(l)}\|_2^2.$$

Gradients: The optimizer requires $\frac{\partial E}{\partial \omega^{(l)}}$ and $\frac{\partial E}{\partial b^{(l)}}$ for each layer l.

4 Neural Network Architecture and Training Details

4.1 Architecture

A custom MLP class was implemented, including: weight initialization , forward propagation , vectorized loss computation , backpropagation via `compute_gradients` , and all operations are vectorized to avoid explicit for-loops.

4.2 Weight Initialization and Activation Functions

For weight initialization, we employed the Xavier method (Glorot & Bengio, 2010), which is well-suited for Tanh activations, and the He method (He et al., 2015), which is effective for ReLU-like activations and was also applied to Swish. The activation functions used in the network were ReLU, Tanh, and Swish. To mitigate numerical instabilities, we applied gradient clipping and pre-activation clipping.

4.3 Backpropagation Derivation

Output Layer Error Signal (with MSE defined as $\frac{1}{N}\sum(y-\hat{y})^2$):

$$\delta^{(L)} = \frac{\partial E}{\partial \hat{y}} = \frac{2}{N}(\hat{y} - y).$$

Hidden Layers:

For hidden layer l,

$$\delta^{(l)} = \left(\delta^{(l+1)}\, W^{(l+1)\top}\right) \odot f'(z^{(l)}),$$

where $\delta^{(l+1)}$ is the error from layer $(l+1)$, $W^{(l+1)\top}$ is the transpose of the next layer's weight matrix, and $f'(z^{(l)})$ is the derivative of the activation function at layer l.

Gradients w.r.t. Weights and Biases:

$$\nabla_{\omega^{(l)}} E = a^{(l-1)\top}\, \delta^{(l)} + 2\,\lambda\,\omega^{(l)},$$

$$\nabla_{b^{(l)}} E = \sum_{i=1}^{N} \delta_i^{(l)}.$$

Since $\frac{1}{N}$ is included in the MSE definition, no extra $\frac{1}{N}$ factor is necessary here.

5 Hyperparameter Tuning

We conducted a grid search with 5-fold cross-validation. The search space included:

- **Hidden layer sizes**: [32,16], [64,32], [128,64], [64,32,32], [128,64,32], [128,64,32,16]
- **Regularization** (λ): {0.001, 0.01, 0.1}
- **Activation functions**: {ReLU, Tanh, Swish}

- **Initialization**: {He, Xavier}

Best Hyperparameters based on cross-validation MAPE and final regularized loss:

- Number of layers: 2
- Hidden Layer Sizes: [128, 64]
- Regularization Term: $\lambda = 0.01$
- Activation Function: `tanh`
- Initialization: `He`

6 Optimization Details

- **Optimizer**: L-BFGS-B
- **Stopping Criteria**: Function tolerance = 10^{-4}, max iterations = 200
- **Convergence for the test set**:
  - Iterations: 50
  - Initial Loss: 1681.80
  - Final Loss: 91.31
  - Message: `CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH`
  - `result.success = True`

7 Final Model Training and Results

- **Final Training**: Retrained with L-BFGS-B on the best hyperparameters with the following performance:
  - Initial regularized error (train): 1681.8002
  - Final regularized error (train): 99.0743
  - Validation error (avg k-fold): 99.9848
  - Test error: 91.3135
  - Initial MAPE (train avg): 99.7089
  - Final MAPE (train avg): 22.9325
  - Final MAPE (validation): 23.0615
  - Final MAPE (test): 23.5604

Table 1: Performance Table

Final Train Loss	Final Test Loss	Final Train MAPE	Final Test MAPE	Optimization Time(s)
99.0743	91.3135	22.9325	23.5604	19.0903

Table 2: Settings Table

Hidden Layer 1		Hidden Layer 2		Other
Number of Neurons	Nonlinearity	Number of Neurons	Nonlinearity	Regularization Term
128	Hyperbolic tangent	64	Hyperbolic tangent	L2

8 Conclusion

In this project, we developed and trained a Multi-Layer Perceptron (MLP) to predict age using a given set of input features. The model's performance was optimized via the L-BFGS-B algorithm, converging from an initial loss of 1681.80 down to 98.77 within 92 iterations. Using 5-fold cross-validation to guide hyperparameter selection, the best model achieved a final training MAPE of 22.975 and a validation MAPE of 23.118 (numbers subject to slight variation upon re-runs).

A key observation throughout this process was the trade-off between computational requirements and convergence precision. The success of L-BFGS-B depends heavily on both the maximum number of iterations and the function tolerance. For practical runtime considerations, these were set to reasonable defaults (maximum iterations = 200, tolerance = 10^{-4}). If better precision and lower residual errors are necessary, one could increase the maximum iterations and reduce the tolerance to allow deeper exploration of the loss landscape—albeit at the expense of increased computation time. Balancing these factors is crucial to achieving both feasible training times and accurate results.

References

- Glorot, X., & Bengio, Y. (2010). *Understanding the difficulty of training deep feedforward neural networks.*
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.*
- Ramachandran, P., Zoph, B., & Le, Q.V. (2017). *Searching for Activation Functions.*