Neural Network Implementation for Regression Analysis: TensorFlow-Based Parameter Optimization

Ahnaf Rahman

November 2025

Technical Environment

Programming Language: Python 3.13.2

Operating System: Ubuntu Linux 6.8.0-38-generic, x86_64

Hardware Configuration:

- CPU: 13th Gen Intel Core i5-13400F, 10 cores, 16 threads
- GPU: NVIDIA GeForce RTX 3060, 8 GB VRAM, CUDA 12.2

Software Libraries:

- TensorFlow Version: 2.20.0
- Pandas: for dataframe import and manipulation
- Scikit-learn: for data preprocessing
- Random: for reproducibility seed setting

Methodology Overview

This implementation utilizes TensorFlow, a dedicated machine learning library, to construct and optimize a neural network for regression analysis. TensorFlow's high-level APIs significantly simplify the neural network construction process by automatically handling feedforward propagation and backpropagation mechanisms.

Key TensorFlow Components:

- `Sequential`: Facilitates layer construction, input shape definition, and activation function specification
- `SGD (Stochastic Gradient Descent)`: Optimizer using fixed learning rate for weight and bias updates
- `Adam`: Adaptive optimizer with dynamic learning rate adjustment for faster convergence

1 Data Import and Preprocessing

1.1 Dataset Import

The dataset was imported using pandas with space delimiters for column separation. The identifier column was removed as irrelevant for model training. Features Con_1, Con_2, Con_3, and Con_4 were separated into variable X, while the target variable (rejects) was assigned to y.

1.2 Train-Test-Validation Split

A three-way data split was implemented using scikit-learn's `train_test_split` function:

- Training set: 70%
- Validation set: 15%
- Test set: 15%

The data was first split into 70% training and 30% remainder, then the remainder was evenly divided into validation and test sets.

Rationale: A simple 80/20 train-test split suffers from data leakage when hyperparameters are tuned based on test performance. The model becomes biased toward the test set, compromising generalizability assessment. Additionally, without a validation split, there is no effective mechanism to monitor overfitting or underfitting during training.

1.3 Feature Scaling

StandardScaler from scikit-learn was applied to normalize all feature sets. The scaler was fitted on training data and subsequently applied to transform validation and test sets, preventing data leakage. Random seed was set to 42 for reproducibility.

2 Neural Network Architecture Design

2.1 Layer Construction

The neural network architecture was implemented using TensorFlow's Sequential API:

Listing 1: Hidden Layer Configuration Grid Search

```
hidden_layer_configs = [
    [32, 16],
    [64, 32],
    [128, 64],
    [256, 128],
    [512, 256],
    [1024, 512]
]

for config in hidden_layer_configs:
    print(f"\nTraining model with hidden layers: {config}")

    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Dense(config[0],
                                    activation='relu',
                                    input_shape=(4,)))
    for neurons in config[1:]:
```

```
        model.add(tf.keras.layers.Dense(neurons,
                                        activation='relu'))
    model.add(tf.keras.layers.Dense(1))
```

The architecture consists of:

- Input layer: 4 neurons (implicitly created via `input_shape=(4,)`)
- Hidden layer 1: Variable units defined by `config[0]`
- Hidden layer 2: Variable units defined by `config[1]`
- Output layer: 1 neuron (regression output)

ReLU (Rectified Linear Unit) activation was selected for hidden layers to introduce non-linearity essential for learning complex patterns.

2.2 Model Compilation

Listing 2: Model Compilation Configuration

```
model.compile(
    optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate,
                                      momentum=0.9),
    loss='mean_squared_error',
    metrics=['mean_absolute_error']
)
```

Configuration Details:

- Optimizer: SGD with momentum of 0.9 to accelerate convergence
- Loss function: Mean Squared Error (MSE)
- Additional metrics: Mean Absolute Error (MAE)

2.3 Model Training

Listing 3: Model Fitting Process

```
history = model.fit(
    X_train_scaled, y_train,
    epochs=epochs,
    validation_data=(X_val_scaled, y_val),
    verbose=0
)
```

Training history was captured for subsequent analysis of learning curves and performance metrics.

3 Hyperparameter Optimization Strategy

3.1 Hidden Layer Architecture Grid Search

A systematic grid search was conducted over six architectural configurations. The design principle of halving neurons across layers forces information compression, improving computational efficiency while maintaining representational capacity.

Tested configurations: [32,16], [64,32], [128,64], [256,128], [512,256], [1024,512]

3.2 Epoch and Learning Rate Tuning

Initial Configuration:

- Epochs: 100
- Learning rate: 0.01

The number of epochs was progressively increased to allow the model more opportunities to capture complex relationships in the data.

Improved Configuration:

- Epochs: 300
- Learning rate: 0.01

Result: Significant MSE improvement. Configuration [256,128] showed strong validation MSE of 0.000693 without overfitting indicators.

Further testing at 400, 500, and 600 epochs showed minimal validation MSE improvement, with increasing overfitting tendency (large gap between training and validation MSE at 500 epochs).

Optimal Configuration:

- Epochs: 300
- Learning rate: 0.1

Result: Validation MSE decreased significantly. Configuration [512,256] achieved lowest validation MSE with minimal overfitting.

Suboptimal Configuration (learning rate too high):

- Epochs: 300
- Learning rate: 0.3

Result: Validation MSE increased, indicating the learning rate of 0.3 negatively impacted performance.

3.3 Activation Function Selection

ReLU vs. Sigmoid Comparison:

ReLU was selected as the optimal activation function for hidden layers because:

- The target variable (rejects) is continuous
- Sigmoid compresses values to [0,1], inhibiting convergence and limiting value representation
- Tanh compresses values to [-1,1], causing similar issues
- ReLU allows any non-negative real number, enabling efficient gradient updates and faster convergence

Experimental Validation: Testing with Sigmoid activation (300 epochs, learning rate 0.1) resulted in significantly higher validation MSE, confirming ReLU superiority for this regression task.

3.4 Optimizer Comparison: SGD vs. Adam

Adam Testing:

Adam optimizer was tested with learning rates of 0.1, 0.01, and 0.001. The optimal configuration (learning rate 0.001) produced strong validation MSE but still underperformed compared to SGD.

Analysis: Adam's adaptive learning rate may cause premature convergence to suboptimal local minima, explaining SGD's superior performance in this application.

4 Final Model Configuration

Optimal Hyperparameters:

- Hidden layer configuration: [512, 256]
- Learning rate: 0.1
- Epochs: 300
- Optimizer: SGD with momentum 0.9
- Activation function: ReLU

Validation Performance:

- Validation MSE: 0.00028
- Validation MAE: 0.013625

Listing 4: Final Model Implementation

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(512, activation='relu',
                                input_shape=(4,)))
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dense(1))

model.compile(
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.1,
                                      momentum=0.9),
    loss='mean_squared_error',
    metrics=['mean_absolute_error']
)

history = model.fit(
    X_train_scaled, y_train,
    epochs=300,
    validation_data=(X_val_scaled, y_val),
    verbose=0
)

test_loss, test_mae = model.evaluate(X_test_scaled, y_test,
                                     verbose=0)
```

Test Performance:

- Test MSE: 0.000275
- Test MAE: 0.012691

The close alignment between validation and test metrics indicates strong model generalizability to unseen data.

5 Analysis and Discussion

5.1 Parameter Sensitivity

Different hyperparameter configurations lead to different optimization trajectories. Key findings:

- **Epochs:** Increasing from 100 to 300 significantly improved performance. Further increases (400-600) yielded diminishing returns with overfitting risk.
- **Learning rate:** A balanced learning rate of 0.1 was optimal. Lower rates (0.01) risked convergence to suboptimal local minima; higher rates (0.3) caused optimization instability.
- **Architecture:** Deeper networks [512,256] captured complex relationships better than shallow configurations, without significant overfitting when properly regularized.
- **Optimizer:** SGD with momentum outperformed Adam, likely due to Adam's potential for premature convergence in this problem space.

While different parameter combinations can potentially yield similar solutions (especially with small learning rates allowing convergence to similar weight spaces), exact parameter convergence is rare.

5.2 Justification for Final Configuration

Grid Search: Systematic exploration of architectural configurations identified [512,256] as optimal when combined with appropriate learning rate and epoch count.

Iterative Refinement: Starting from conservative baselines (100 epochs, 0.01 learning rate), parameters were incrementally adjusted based on validation performance. The 300-epoch threshold balanced learning capacity with overfitting prevention.

Activation Function: ReLU's unbounded positive range and computational efficiency made it ideal for continuous regression targets, validated by empirical comparison with Sigmoid.

Optimizer Selection: SGD with momentum provided more stable convergence than Adam for this dataset, achieving superior validation metrics.

Learning Rate Balance: The 0.1 learning rate successfully navigated between:

- Too low: Suboptimal local minima convergence
- Too high: Gradient instability and overshooting optimal solutions

6 Final Architecture Summary

Network Structure:

- Input layer: 4 neurons
- Hidden layer 1: 512 units (ReLU activation)
- Hidden layer 2: 256 units (ReLU activation)
- Output layer: 1 neuron (linear activation)

Training Configuration:

- Optimizer: SGD (momentum=0.9)

- Learning rate: 0.1
- Epochs: 300
- Batch size: Default (32)

Final Performance Metrics:

- Test MSE: 0.000275
- Test MAE: 0.012691
- Training SSE: 0.010920
- Training MSE: 0.000052
- Test SSE: 0.012358

7 Visualizations

7.1 Learning Curves

[Placeholder for separated plots showing training and validation MSE/MAE over epochs]

7.2 Combined Performance Plot

[Placeholder for combined visualization of model performance metrics]

8 Conclusion

This implementation demonstrates a systematic approach to neural network hyperparameter optimization for regression tasks. Through rigorous grid search and iterative refinement, the final model achieved strong generalization performance (Test MSE: 0.000275) while avoiding overfitting.

Key success factors included:

1. Proper three-way data splitting to prevent evaluation bias
2. Systematic grid search over architectural configurations
3. Careful learning rate selection balancing convergence speed and stability
4. Appropriate activation function choice for continuous targets
5. Empirical validation of optimizer performance

The close alignment between validation and test metrics validates the model's readiness for deployment on similar regression tasks.