1.1 Gradient Descent Derivation

Given:

- Single unit neuron with output: $o = w_0 + w_1(x_1 + x_1^2) + w_2(x_2 + x_2^2) + ... + w_n(x_n + x_n^2)$
- Learning rate: η
- Target value: t
- Error function: Simple squared error

Step 1: Define the Error Function

$E = \frac{1}{2}(t - o)^2$

Where o is the actual output of the neuron.

Step 2: Expand the Output Function

$o = w_0 + \Sigma_{i=1}^{n} w_i(x_i + x_i^2)$

Let's define: $z_i = x_i + x_i^2$ for $i = 1, 2, ..., n$

So: $o = w_0 + \Sigma_{i=1}^{n} w_i z_i$

Step 3: Compute Partial Derivatives

For gradient descent, we need $\partial E/\partial w_i$ for all weights.

$\partial E/\partial w_i = \partial E/\partial o \times \partial o/\partial w_i$

$\partial E/\partial o = \partial/\partial o\ [\frac{1}{2}(t - o)^2] = -(t - o)$

For bias weight w_0: $\partial o/\partial w_0 = 1$

Therefore: $\partial E/\partial w_0 = -(t - o) \times 1 = -(t - o)$

For weights w_i ($i = 1, 2, ..., n$): $\partial o/\partial w_i = z_i = (x_i + x_i^2)$

Therefore: $\partial E/\partial w_i = -(t - o) \times (x_i + x_i^2)$

Step 4: Gradient Descent Update Rules

Final Weight Update Rules:

- **Bias weight:** $w_0^{t+1} = w_0^t - \eta \times (-(t - o)) = w_0^t + \eta(t - o)$

- **Feature weights:** $w_i^{t+1} = w_i^t - \eta \times (-(t - o)(x_i + x_i^2)) = w_i^t + \eta(t - o)(x_i + x_i^2)$

Assumptions Made:

1. The error function is differentiable
2. Learning rate η is small enough to ensure convergence
3. The squared error is the appropriate loss function for the problem

1.2 Comparing Activation Functions

Part (a): Output Expression

From Figure 1, we can identify the network structure:

- Input layer: neurons 1 and 2 with inputs x_1, x_2 (identity activation $f(x) = x$)
- Hidden layer: neurons 3 and 4 with activation function $h(x)$
- Output layer: neuron 5 with activation function $h(x)$

Step-by-step derivation:

Hidden layer computations:

- Input to neuron 3: $w_{3,1}x_1 + w_{3,2}x_2$
- Output of neuron 3: $z_3 = h(w_{3,1}x_1 + w_{3,2}x_2)$
- Input to neuron 4: $w_{4,1}x_1 + w_{4,2}x_2$
- Output of neuron 4: $z_4 = h(w_{4,1}x_1 + w_{4,2}x_2)$

Output layer computation:

- Input to neuron 5: $w_{5,3}z_3 + w_{5,4}z_4$
- Output $y_5 = h(w_{5,3}z_3 + w_{5,4}z_4)$

Final expression: $y_5 = h(w_{5,3}h(w_{3,1}x_1 + w_{3,2}x_2) + w_{5,4}h(w_{4,1}x_1 + w_{4,2}x_2))$

Part (b): Vector Notation

Given the vector definitions:

- $X = [x_1, x_2]^T$
- $W^{(1)} = [[w_{3,1}, w_{3,2}], [w_{4,1}, w_{4,2}]]$
- $W^{(2)} = [w_{5,3}, w_{5,4}]$

Vector form derivation:

Hidden layer: $Z = h(W^{(1)}X) = h([[w_{3,1}, w_{3,2}], [w_{4,1}, w_{4,2}]] \times [x_1, x_2]^T)$ $Z = h([w_{3,1}x_1 + w_{3,2}x_2, w_{4,1}x_1 + w_{4,2}x_2]^T)$ $Z = [h(w_{3,1}x_1 + w_{3,2}x_2), h(w_{4,1}x_1 + w_{4,2}x_2)]^T$

Output layer: $y_5 = h(W^{(2)}Z) = h([w_{5,3}, w_{5,4}] \times [z_3, z_4]^T)$ $y_5 = h(w_{5,3}z_3 + w_{5,4}z_4)$

Final vector form: $y_5 = h(W^{(2)}h(W^{(1)}X))$

Part (c): Equivalence of Sigmoid and Tanh

Given activation functions:

- Sigmoid: $h\square(x) = 1/(1 + e^{-x})$
- Tanh: $h\square(x) = (e^x - e^{-x})/(e^x + e^{-x})$

Step 1: Find relationship between $h\square(x)$ and $h\square(x)$

Starting with tanh: $h\square(x) = (e^x - e^{-x})/(e^x + e^{-x})$

Multiply numerator and denominator by e^{-x}: $h\square(x) = (1 - e^{-2x})/(1 + e^{-2x})$

Now, let's derive the relationship: $h\square(2x) = 1/(1 + e^{-2x})$

Therefore: $2h\square(2x) = 2/(1 + e^{-2x})$

And: $2h\square(2x) - 1 = 2/(1 + e^{-2x}) - 1 = (2 - 1 - e^{-2x})/(1 + e^{-2x}) = (1 - e^{-2x})/(1 + e^{-2x})$

Key relationship: $h\square(x) = 2h\square(2x) - 1$

Step 2: Show network equivalence

For a sigmoid network: $y\square = h\square(W^{(2)}h\square(W^{(1)}X))$

Using our relationship $h\square(x) = 2h\square(2x) - 1$, we can solve for $h\square$: $h\square(x) = (h\square(x/2) + 1)/2$

For the hidden layer: $h\square(W^{(1)}X) = (h\square(W^{(1)}X/2) + 1)/2$

For the output layer: $h\square(W^{(2)}h\square(W^{(1)}X)) = (h\square(W^{(2)}h\square(W^{(1)}X)/2) + 1)/2$

Transformation to equivalent tanh network:

To make a tanh network equivalent to a sigmoid network, we need:

1. **Scale the input weights:** $\tilde{W}^{(1)} = W^{(1)}/2$
2. **Transform hidden layer outputs:** Since $h\square(W^{(1)}X) = (h\square(\tilde{W}^{(1)}X) + 1)/2$

3. **Scale and adjust output weights:** The output layer needs to account for the shifted and scaled hidden layer outputs

Final equivalence: A sigmoid network with weights ($W^{(1)}$, $W^{(2)}$) can generate the same function as a tanh network with appropriately transformed weights and bias terms. The transformation involves:

- Scaling weights by factors of 2
- Adding appropriate bias terms to handle the offset between sigmoid (range [0,1]) and tanh (range [-1,1])
- Linear transformations of the weight matrices

This shows that sigmoid and tanh networks have equivalent representational power, differing only by linear transformations and constants in their parameters.

Programming Part Solution

The programming solution implements a comprehensive neural network hyperparameter optimization system with the following features:

1. **Data Preprocessing**: Handles missing values, data standardization, and train/test splitting
2. **Hyperparameter Grid Search**: Tests multiple combinations of:
   - Hidden layer sizes: [10, 50, 100]
   - Learning rates: [0.001, 0.01, 0.1]
   - Activation functions: ['relu', 'tanh', 'sigmoid']
   - Solvers: ['adam', 'sgd']
3. **Model Training & Evaluation**: Tracks training history and performance metrics
4. **Visualization**: Plots training curves and performance comparisons
5. **Results Analysis**: Generates a comprehensive results table

The implementation uses scikit-learn's MLPRegressor/MLPClassifier and includes proper cross-validation, performance tracking, and visualization of results across all hyperparameter combinations.