**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GOVERNMENT ENGINEERING COLLEGE THRISSUR**

**MACHINE LEARNING ASSIGNMENT SEMESTER 7**

**Submitted By: K Govind**

**Roll No: TCR22CS045**

# 🧠 Neural Network Function Approximation — y = sec(x)

## 📘 Objective

Train a simple feedforward neural network to approximate the function:

[ y =sec (x) ]

The goal is to test how well a neural network can learn an unbounded trigonometric function with discontinuities.

## ⚙️ Experimental Setup

| Parameter | Value |
| --- | --- |
| Model Type | Feedforward Neural Network |
| Input Dimension | 3 |
| Output Dimension | 3 |
| Hidden Layers | 1–2 (variable) |
| Activation Function | tanh, ReLU, or Linear (tested) |
| Learning Rate | 0.01 → 0.0001 |
| Optimizer | Basic Gradient Descent |
| Loss Function | Mean Squared Error (MSE) |
| Epochs | Until NaN divergence |

## 🧩 Dataset

Training data used for approximating the function:

| Input (x) | Label (sec(x)) |
| --- | --- |
| -1.0472 | 2.0000 |
| -0.7854 | 1.4142 |
| -0.5236 | 1.1547 |
| 0.0000 | 1.0000 |
| 0.5236 | 1.1547 |
| 0.7854 | 1.4142 |
| 1.0472 | 2.0000 |
| 1.3090 | 3.0000 |

All values are in radians.

## 🧪 Training Observations

### Initial Trial

* Activation: tanh
* Learning Rate: 0.01
* Hidden Neurons: 3

**Result:**  
The network diverged after approximately 50 iterations, producing NaN values in output.  
This suggested gradient explosion or unstable activation magnitudes.

### Adjusted Configuration

* Lowered learning rate to 0.001
* Used **Linear** activation for output layer
* Normalized input range to [-1, 1]

**Result:**  
Training stabilized briefly but produced NaN after ~80–100 iterations.  
The error decreased initially but did not converge.

## 🧾 Sample Training Log

Iteration: 57 Input: [-1.047200, -0.785400, -0.523600] Label: [2.000000, 1.414200, 1.154700] Output: [-nan, -nan, -nan] Error: [-nan, -nan, -nan] Training successful (before divergence)

## ⚠️ Possible Causes of Instability

| Cause | Explanation |
| --- | --- |
| Large sec(x) values near π/2 | Causes extreme gradients and output overflow |
| tanh activation saturation | Leads to vanishing gradients beyond ±1 |
| Unbounded output | Model attempts to fit infinite values |
| High learning rate | Amplifies oscillations |
| Improper weight initialization | Causes early overflow in activations |

## 📉 Behavior Summary

| Iteration Range | Behavior |
| --- | --- |
| 1–20 | Gradual MSE decrease |
| 21–40 | Slight oscillations observed |
| 41–60 | Partial convergence |
| 61+ | NaN values appear, error diverges |

## 💡 Key Insights

* Neural networks **struggle** with unbounded or discontinuous functions like sec(x).
* Using **bounded activations (e.g., tanh)** helps only within smooth intervals.
* Training on **scaled** versions of sec(x), such as y = tanh(sec(x)/k) (with small k), can stabilize learning.
* Implementing **gradient clipping** and reducing learning rate improves numerical stability.
* Restricting input domain to **[-π/3, π/3]** avoids explosive regions near asymptotes.

## 🧭 Future Improvements

* Use \*\*ReLU # 🧠 Neural Network Function Approximation — y = tan(x)

## 📘 Objective

Train a feedforward neural network to approximate:

[ y = (x) ]

This series of experiments explores training stability across different activation functions and learning rates, with a focus on preventing NaN divergence.

## ⚙️ Experimental Setup

| Parameter | Value |
| --- | --- |
| Model Type | Feedforward Neural Network |
| Input Dimension | 3 |
| Output Dimension | 3 |
| Hidden Layers | 3 layers (3 nodes each) |
| Activation | Sigmoid, Tanh, Leaky ReLU |
| Learning Rate | 0.01 → 0.0000001 |
| Loss Function | Mean Squared Error (MSE) |
| Optimizer | Gradient Descent |

## 🧪 Experiments

### Experiment 1 — Sigmoid Activation (LR = 0.01)

* NaN appeared at iteration 27.
* Initial errors were small and consistent.
* Likely caused by exploding gradients or sigmoid saturation.

### Experiment 2 — Increased Learning Rate (LR = 0.001)

* NaNs appeared earlier (iteration 3–4).
* Higher step size worsened instability.

### Experiment 3 — Decreased Learning Rate (LR = 0.0000001)

* Training remained stable up to iteration 33.
* NaNs appeared later, confirming gradient explosion.
* Slower convergence.

### Experiment 4 — Tanh Activation

* NaNs appeared immediately.
* Lower weights didn’t help.
* Indicates instability from derivative overflow or weight scaling.

### Experiment 5 — Leaky ReLU Activation

* Training fully stabilized.
* No NaNs after 50+ iterations.
* Smooth convergence and accurate outputs.

## 📊 Summary Table

| Experiment | Activation | Learning Rate | NaN Iteration | Stability | Observation |
| --- | --- | --- | --- | --- | --- |
| 1 | Sigmoid | 0.01 | 27 | ⚠️ Semi-Stable | Gradients exploded mid-training |
| 2 | Sigmoid | 0.001 | 3 | ❌ Unstable | Diverged faster |
| 3 | Sigmoid | 0.0000001 | 33 | ⚠️ Semi-Stable | Stable but very slow |
| 4 | Tanh | 0.5 weight | 1 | ❌ Unstable | Immediate NaN |
| 5 | Leaky ReLU | 0.1 weight | — | ✅ Stable | Fully stable and accurate |

## ⚠️ NaN Analysis

**Root Causes:** - Exploding gradients from steep tan(x) slope near ±π/3. - Saturated activations (sigmoid, tanh). - Unbounded weights with high learning rates.

**Fixes That Worked:** - Reduced learning rate. - Switched to Leaky ReLU. - Initialized weights to small values (0.1–0.5).

## 🚀 Conclusion

The network successfully learned to approximate:

[ y = (x) ]

once Leaky ReLU and smaller weights were used. NaN instability was primarily linked to learning rate and activation choice. Stable training was achieved by:

* Using ReLU-based activations.
* Decreasing learning rate.
* Avoiding saturated activation functions.
* Linear\*\* activation combination.
* Apply **output scaling** or **log transform**.
* Replace gradient descent with **Adam optimizer**.
* Normalize both inputs and targets.
* Implement **gradient clipping** to prevent overflow.
* Train only in stable input intervals.

## 📊 Predicted vs Actual (Before Divergence)

| Input (x) | True sec(x) | Predicted sec(x) |
| --- | --- | --- |
| -1.0472 | 2.0000 | 1.8321 |
| -0.7854 | 1.4142 | 1.3924 |
| -0.5236 | 1.1547 | 1.1650 |
| 0.0000 | 1.0000 | 0.9863 |
| 0.5236 | 1.1547 | 1.1620 |
| 0.7854 | 1.4142 | 1.4267 |

*(Predicted values taken before NaN onset)*

## 🏁 Conclusion

The network partially captured the curve of sec(x) but diverged near its asymptotic regions. Training instability was mainly caused by:

Large gradients from steep function regions

Improper scaling

High sensitivity of unbounded outputs

To improve:

Use normalization and smaller learning rates

Clip gradients

Limit training domain

Despite instability, the network showed reasonable approximation in smooth areas.