Neural Networks: A High-Dimensional Application of Approximation Theory

Yahya Izadi

2025-11-21

Table of contents

3. The Existence Theorem

Is there a guarantee that this specific form can approximate the complex function of handwriting?

> **Note**
>
> [!note] Universal Approximation Theorem (Cybenko, 1989)
>
> The set of functions generated by neural networks with sigmoidal activation functions is **dense** in $C(K)$.
>
> Mathematically:
>
> $$\overline{\text{span}\{\sigma(\mathbf{w}^T\mathbf{x}+b)\}} = C(K)$$

This theorem is the direct generalization of the Weierstrass theorem. It assures us that a solution exists; the challenge lies only in *finding* the coefficients.

---

4. The Least Squares Problem

To find these coefficients, we perform exactly the same operation we studied in "Best Approximation in L^2 Norm." Since we define the error over discrete data points, we use the **Discrete ℓ^2 Norm**:

$$E(\theta) = \|f - g\|_2^2 = \sum_{i=1}^{M} (y_i - g(\mathbf{x}_i; \theta))^2$$

$$

In Data Science, minimizing this norm is called **"Training."** The key difference from our class examples is that the dependence on parameters $\mathbf{w}$ is **non-linear**, necessitating iterative numerical methods like Gradient Descent instead of direct linear solvers.

---

5. Stability and the Runge Phenomenon

In Approximation Theory, we learned that increasing the degree of approximation is not always beneficial. High-degree polynomials can fit the data points perfectly but oscillate wildly between them.

In Data Science, this is called **Overfitting**. The simulation below demonstrates this concept:

```python
import numpy as np
import matplotlib.pyplot as plt

# Setup
np.random.seed(42)
x = np.linspace(0, 10, 15)
y_true = np.sin(x) + 0.5 * x  # True function
y_noise = y_true + np.random.normal(0, 0.3, len(x)) # Noisy observations

x_plot = np.linspace(0, 10, 200)

# 1. High Degree Approximation (Simulating Runge Phenomenon/Overfitting)
# Degree 14 passes through all 15 points perfectly
coeffs_high = np.polyfit(x, y_noise, 14)
p_high = np.poly1d(coeffs_high)

# 2. Smooth Approximation (Generalization)
# Degree 3 is more robust
```

```python
coeffs_low = np.polyfit(x, y_noise, 3)
p_low = np.poly1d(coeffs_low)

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(x, y_noise, color='red', s=50, label='Observed Data (Noisy)', zorder=5)
plt.plot(x_plot, np.sin(x_plot) + 0.5 * x_plot, 'k--', alpha=0.3, label='True Function f(x)'
plt.plot(x_plot, p_high(x_plot), color='blue', alpha=0.6, label='High Degree (Unstable / Ove
plt.plot(x_plot, p_low(x_plot), color='green', linewidth=2, label='Smooth Approx (Stable)')

plt.ylim(-2, 8)
plt.title('Accuracy vs. Stability: Runge Phenomenon in Data Science')
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

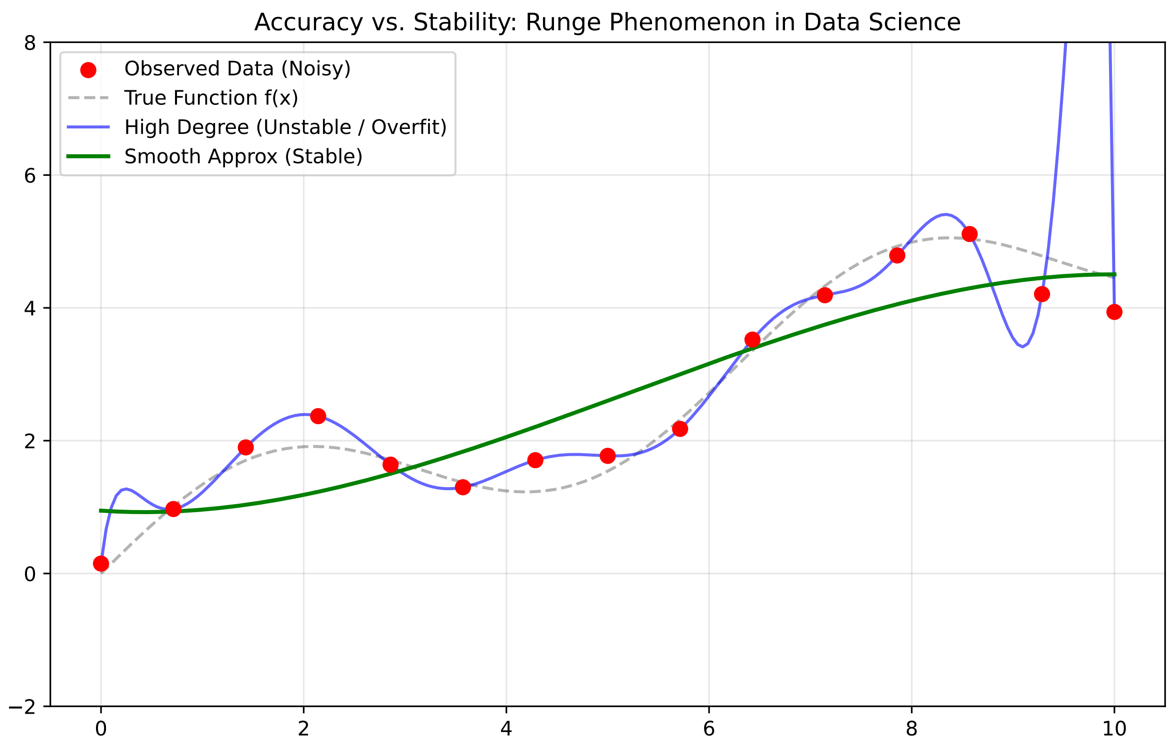

Figure 1: Runge Phenomenon vs. Smooth Approximation

The Mathematical Solution: Regularization

To prevent these oscillations, we add a penalty term to the error function, known as **Tikhonov Regularization**:

$$\min \left(\|f - g\|_2^2 + \lambda \|\theta\|_2^2 \right)$$

This forces the approximating function to remain “smooth” (small derivatives) and filters out the high-frequency oscillations caused by noise.

Conclusion

Handwriting recognition—or any Machine Learning task—is not magic. It is fundamentally a **Function Approximation Problem** with three distinct characteristics:

1. The vector space is significantly larger ($\mathbb{R}^{784}$).
2. The basis functions have changed from Polynomials to Sigmoids.
3. We use discrete norms and numerical optimization to find coefficients.

Thus, a successful Data Scientist is, in essence, an Applied Mathematician skilled in high-dimensional approximation.