Report: The Neural Network Lemma

September 22, 2025

1 Introduction

In our calculus classes, we all learned that tools like the Taylor series can approximate a function very well. However, as you will see, a neural network can learn to approximate several different functions at once, and it can even replicate their derivatives with remarkable accuracy.

2 Part 1: Lemma 3.1

2.1 What is this Lemma saying?

In simple terms, Lemma 3.1 presents four key points:

- **Extremely Simple Tools:** We only need a structurally simple "shallow" neural network (you can think of it as a small company with just one layer of employees).
- **Powerful Approximation Ability:** This network is excellent at approximating specific functions, namely those with odd powers $(x, x^3, x^5, \dots)$.
- **Approximating Derivatives:** This approximation isn't just about the function values looking similar. It also replicates the function's "dynamics," such as where it increases fastest or has inflection points (which are things derivatives tell us). The intimidating mathematical notation in the report, $\|\cdot\|_{W^{k,\infty}}$, can be understood as an "all-around similarity score" that checks the similarity of the function itself and its various derivatives.
- **The Cost:** To achieve high precision (a very small error ϵ) or to approximate more functions simultaneously, the network's parameter values (weights) will become absurdly large. It's like trying to calculate all the secrets of the universe on a small calculator—the calculator would overheat and burn out first.

2.2 Where does the proof idea come from? Thinking with Calculus

The coolest part of this proof is that it constructively "builds" the network for you. And the core idea behind it is very similar to what we learned in calculus.

Core Idea: Using a "left-minus-right" trick to filter out what we want

In calculus textbooks, the derivative is approximated like this:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

This technique of subtracting the value at (x-h) from the value at (x+h) and dividing by the distance is called "finite difference." The authors discovered that by cleverly shifting, adding, and subtracting the $\tanh(x)$ function (which is what a neural network does), one can create a sieve that filters out the unwanted terms in a Taylor expansion, leaving only the desired x^p.

You can think of it this way:

- The $\tanh(x)$ function inherently contains all odd powers: $x, x^3, x^5, \dots$.
- The complex mathematical expression in the proof is essentially a super-filter.

- It combines values of tanh at different points in such a way that all powers lower than x^p are cancelled out, and powers higher than x^p become negligible. In the end, what the network outputs is a very "pure" x^p.

The h in the proof acts like a precision knob. The smaller the h, the better the approximation. This also explains why, when we require the error $\epsilon \to 0$, h must also become smaller, causing the weights (which contain terms like $1/h^p$) to explode.

3 Part 2: Lemma 3.2

After Lemma 3.1 gave us the tools to create odd powers, Lemma 3.2 shows us how to use these tools to generate all powers, including the even ones.

3.1 What is this lemma saying?

This lemma basically states:

- The network can now approximate all polynomial functions up to a degree of s (regardless of odd or even powers).
- To further generate all powers, including the even ones, the network's size (width) needs to be tripled.
- The weight parameters will still grow to be enormous.

3.2 The Core Idea

The clever idea here uses mathematical induction to guarantee success.

Core Idea: If you can make a cubic, you can make a quadratic

The key is an algebraic trick. For example, to get y^2, you can construct it from cubic terms:

$$y^2 = \frac{1}{6}\left((y+1)^3 - (y-1)^3 - 2\right)$$

This tells us that by combining a few odd powers, we can create an even power. Lemma 3.2 turns this idea into a procedure for constructing any even power x^{2n}. The ingredients are the odd powers we can already create and the lower-order even powers we have already constructed in previous steps.

Why does the network need to be 3 times wider?

Because this procedure requires three main ingredients: the original odd-power generator $\hat{f}(y)$, and its shifted versions, $\hat{f}(y+\alpha)$ and $\hat{f}(y-\alpha)$.

The proof uses mathematical induction to ensure the error doesn't get out of control as we construct higher and higher powers.

- First, they prove that constructing x^2 this way is feasible and the error can be well-controlled.
- Then, they show that if we can successfully construct all even powers up to $x^{2(n-1)}$, this method guarantees the successful construction of the next one, x^{2n}. This ensures the procedure works for all even powers!

4 Conclusion

Together, these two theorems reveal some fundamental properties of neural networks:

- **Powerful Approximation Capability:** Even the simplest "shallow" networks are capable of excellent approximation. Their ability even surpasses that of traditional polynomials because they can also learn the derivatives of functions.

- **The Proof Also Reveals a Harsh Reality: Parameter Explosion.** In theory, they can approximate perfectly, but in practice, this may be impossible because it would require astronomically large parameter settings.
- **Why Deep Learning?:** This problem of parameter explosion also indirectly explains why the community shifted towards "deep" networks (with many layers). It is conjectured that by building deeper networks, one might learn similarly complex functions using more practical, reasonably-sized parameters, thus overcoming the parameter explosion dilemma of shallow networks.

Question on ReLU and Approximation Theory

The derivation of approximation theory seems to be highly dependent on the smoothness and differentiability of the activation function $\sigma(x)$, even requiring its second derivative $\sigma''(b)$ to exist and be non-zero. However, many contemporary neural networks use activation functions like ReLU, which is non-differentiable at the origin. Is this theoretical framework, based on Taylor expansions and finite differences, still applicable to explain the capabilities of ReLU networks?

References

[1] OpenAI. (2024, May). *ChatGPT Language Model.* Personal communication. (Use ChatGPT to understand the details of neural networks).

[2] T. De Ryck, S. Lanthaler, and S. Mishra. On the approximation of functions by tanh neural networks. *Neural Networks*, 143:732–750, 2021.