Lemma 3.1

Let $k \in \mathbb{N}_0$ and $s \in 2\mathbb{N} - 1$. Then it holds that for all $\varepsilon > 0$ there exists a shallow tanh neural network $\Psi_{s,\varepsilon}: [-M, M] \to \mathbb{R}^{\frac{s+1}{2}}$ of width $\frac{s+1}{2}$ such that

$$\max_{\substack{p \le s \\ p \text{ odd}}} \left\| f_p - (\Psi_{s,\varepsilon})_{\frac{p+1}{2}} \right\|_{W^{k,\infty}} \le \varepsilon$$

Moreover, the weights of $\Psi_{s,\varepsilon}$ scale as

$$O\left(\varepsilon^{\frac{-s}{2}} (2(s+2)\sqrt{2M})^{s(s+3)}\right)$$

for small ε and large s.

We want to construct a ***very shallow*** tanh neural network $\Psi_{s,\varepsilon}$ that can simultaneously approximate several odd-degree monomials

$$f_p(x) = a_p x^p \quad, p = 1, 3, 5, \dots, s \text{ ◦}$$

with high accuracy—not only matching the function values, but also ensuring that the first derivative, second derivative, ..., up to the k-th derivative are all close.

Recall that $\tanh(x)$ can be expanded around x=0:

$$\tanh(x) = x - \frac{1}{3}x^3 + \frac{2}{15}x^5 - \frac{17}{315}x^7 + \cdots$$

That is, $\tanh(x)$ is itself an infinite series of odd-degree terms.

Now consider a neuron output of the form:

$$\sum_{i=1}^{N} w_i \tanh(u_i x + b_i)$$

This is a linear combination of multiple tanh functions.

Since each $w_i \tanh(u_i x + b_i)$ expands into odd powers of x, we can choose the parameters w_i, u_i, b_i so that higher-order terms cancel out, leaving only the monomial x^p.

In other words, by carefully weighting and combining, we can “isolate” a desired power term.

Example (approximating x^5):

Define

$$g(x) = \frac{5}{16}\tanh(x) - \frac{1}{4}\tanh(2x) + \frac{1}{16}\tanh(3x).$$

Expanding each term up to order x^5:

$$\tanh(ux) = \mathrm{u}x - \frac{u^3}{3}x^3 + \frac{2u^5}{15}x^5 + O(x^7).$$

Summing coefficients, we get:

- The coefficient of x:$1 \cdot \left(\frac{5}{16} \cdot 1 - \frac{1}{4} \cdot 2 + \frac{1}{16} \cdot 3\right) = 0 \ \ (\text{cancels to } 0)$
- The coefficient of x^3:$\left(-\frac{1}{3}\right) \cdot \left(\frac{5}{16} \cdot 1^3 - \frac{1}{4} \cdot 2^3 + \frac{1}{16} \cdot 3^3\right) = 0 \ \ (\text{cancels to } 0)$
- The coefficient of x^5:$\left(\frac{2}{15}\right) \cdot \left(\frac{5}{16} \cdot 1^5 - \frac{1}{4} \cdot 2^5 + \frac{1}{16} \cdot 3^5\right) = \frac{2}{15} \cdot \frac{120}{16} = 1$

So the expansion is

$$g(x) = x^5 + C_7x^7 + C_9x^9 + \cdots$$

which means that for small x, $g(x) \approx x^5$.

In our construction, we want one single neural network to approximate ***all*** functions $f_p(x)$ at once. Therefore, we design the network output to be a vector:

$$\Psi_{s,\varepsilon}(x) \in \mathbb{R}^{\frac{s+1}{2}}$$

where the j-th component corresponds to x^{2j-1}. This way, the network simultaneously captures all odd powers up to s.

Since we want not only function values but also derivatives to match, the error is measured in the $W^{k,\infty}$ norm, which controls the maximum deviation up to the k-th derivative:

- $k = 0$: compares function values
- $k = 1$: compares function values and first derivative
- $k = 2$: compares up to the second derivative
- etc.

This ensures that the *entire functional behavior* is approximated, not just pointwise values.

Finally, the author estimates how large the network weights must be in order to achieve this precision:

$$O\left(\varepsilon^{\frac{-s}{2}}(2(s+2)\sqrt{2M})^{s(s+3)}\right)$$

Although this bound grows extremely fast as s increases or as ε becomes small, the key result is that **such a network exists**.

Lemma 3.2

Let $k \in \mathbb{N}_0$ and $s \in 2\mathbb{N}-1$ and $M > 0$. For all $\varepsilon > 0$ there exists a shallow tanh neural network $\Psi_{s,\varepsilon}: [-M, M] \to \mathbb{R}^s$ of width $\frac{3(s+1)}{2}$ such that

$$\max_{p \le s} \left\| f_p - (\Psi_{s,\varepsilon})_p \right\|_{W^{k,\infty}} \le \varepsilon$$

Moreover, the weights of $\Psi_{s,\varepsilon}$ scale as $O\left(\varepsilon^{\frac{-s}{2}}((s+2)\sqrt{M})^{\frac{3s(s+3)}{2}}\right)$ for small ε and large s.

We want to use a **shallow tanh neural network** to simultaneously approximate multiple polynomials $x, x^2, x^3, \dots, x^s$.

According to Lemma 3.1, it is natural to use the neural network to generate odd-degree polynomials, such as $x, x^3, x^5, \dots$.

For even-degree polynomials $x^2, x^4, \dots$, we can use the following technique:

$$y^{2n} = \frac{1}{2\alpha(2n+1)}\left((y+\alpha)^{2n+1} - (y-\alpha)^{2n+1} - 2\sum_{k=0}^{n-1}\binom{2n+1}{2k}\alpha^{2(n-k)+1}y^{2k}\right)$$

This formula shows that even powers can be expressed as a "difference of odd powers" plus a correction term.

Intuitively,$(y+\alpha)^{2n+1} - (y-\alpha)^{2n+1}$ generates many odd-degree terms, and by subtracting the extra lower-degree terms and multiplying by a coefficient, we can precisely isolate the even-degree term. In a neural network, this corresponds to selecting several $\tanh(ux+b)$ neurons (with the shift b corresponding to α) and combining them linearly to approximate even-degree polynomials.

Higher-degree terms beyond s introduce residuals, but by scaling the hidden neuron parameters u_i, we can reduce their influence over the interval $[-M, M]$ to be arbitrarily small. This ensures that both the function values and derivatives are within the desired error $\le \varepsilon$ 。

If we want to accurately approximate a specific polynomial x^p (whether odd or even degree), a single $\tanh(ux+b)$ generates many powers, so we cannot isolate x^p with just one neuron. At least three neurons are needed, combined linearly, to cancel out the undesired powers:

- Neuron 1：provides the positive x^p term
- Neuron 2：cancels the extra lower-degree terms
- Neuron 3：adjusts higher-degree terms or fine-tunes

Therefore, each target polynomial roughly requires 3 neurons, making the total width $\frac{3(s+1)}{2}$ 。

To achieve finer approximation or for higher-degree polynomials, the output weights will grow larger, but the author provides the following upper bound:

$$O\left(\varepsilon^{\frac{-s}{2}}((s+2)\sqrt{M})^{\frac{3s(s+3)}{2}}\right)$$

The key point is that this proves the **existence of a shallow tanh neural network** that can simultaneously approximate all the polynomials, with derivatives also matching accurately.