1. Introduction of lemma 3.1

Given three setups :

- The interval is fixed to [-M, M]
- Given an integer k ≥ 0 (Controlling the k-th order derivative.)
- Given an odd upper bound s (We want to approximate all odd powers x^p simultaneously, where p is less than or equal to s and p is an odd number.)

We will get the following conclusion :

For arbitrary precision ε > 0, there exists a one-hidden-layer tanh neural network $\Psi_{s,\epsilon}$ whose hidden width is (s+1)/2, such that its output vector

$$\Psi_{s,\epsilon}(x) = \big((\Psi_{s,\epsilon})1(x),\, (\Psi s,\epsilon)2(x),\, \ldots,\, (\Psi s,\epsilon)(s+1)/2(x)\big)$$

has the (p+1)/2-th coordinate approximating $f_p(x) = x^p$ with

$$\| f_p - (\Psi s,\epsilon)(p+1)/2 \|_{W^{k,\infty}([-M,M])} \;\le\; \epsilon\,, \quad \text{for every odd } p \le s.$$

Here

$$\|g\|_{W^{k,\infty}([-M,M])} := \max_{0\le m\le k} \Big(\sup_{|x|\le M} |g^{(m)}(x)| \Big)$$

is the Sobolev norm. That is, the maximum error over the function and all derivatives up to order k.

Moreover, the lemma gives an upper bound on the size of the weights needed to reach error ε, the weights grow at most like

$$O\Big(\varepsilon^{-s/2}\,\big(2(s+2)\sqrt{2M}\big)^{\,s(s+3)}\Big)\,, \quad \text{as } \epsilon \to 0 \text{ and for large s}$$

2. Some details of the conclusion

- ## Why we can construct a neural network with hidden width only (s+1)/2 :

  To approximate the largest odd power s, we need slopes {0, ±h, ±2h, …, ±(s/2)h}. Because tanh is an odd function, the pair ±a can share the same hidden neuron with opposite output weights. Thus we keep only the nonnegative slopes plus the center 0, totaling (s+1)/2 neurons. This set of neurons already covers all slopes required for any smaller odd p ≤ s, so a single hidden layer serves all targets simultaneously.

- ## Why the (p+1)/2-th coordinate approximates $f_p(x) = x^p$:

  That coordinate is defined by the centered finite-difference-style linear combination

  $$(\Psi_{s,\varepsilon})(p+1)/2(x) \;=\; \frac{1}{h^p\,\sigma^{(p)}(0)} \sum i = 0^p (-1)^i \binom{p}{i}\, \sigma\Big(\big(\tfrac{p}{2} - i\big)h\,x\Big).$$

  Using a Taylor expansion around x=0 and the moment-cancellation identity, all terms below order p and the (p+1)-st term cancel, leaving precisely the x^p term; the remainder is O(h^2). Choosing h sufficiently small makes the error to ε.

- ## Why is the upper bound of the weight in this form :

  For odd p ≤ s, (p+1)/2-th output is defined as

  $$(\Psi_{s,\varepsilon})_{(p+1)/2}(x) := \frac{1}{h^p\,\sigma^{(p)}(0)} \sum_{i=0}^{p} (-1)^i \binom{p}{i}\, \sigma\Big(\big(\tfrac{p}{2} - i\big)h\,x\Big), \qquad \sigma = \tanh.$$

  Therefore, the output layer weight corresponding to the offset(p/2-i)h is

  $$w_{p,i} = \frac{(-1)^i \binom{p}{i}}{h^p\,\sigma^{(p)}(0)}.$$

Expand σ(ax) at x=0 and use the moment-removing identity. Together with the derivative to the kth order, it can be estimated as

$$\left\| \left(\Psi_{s,\varepsilon}\right)_{(p+1)/2} - x^{p} \right\|_{W^{k,\infty}([-M,M])} \;\le\; C(p,k,M)\, h^{2}.$$

To achieve the error ε, select the step length h, take the worst constants $C_{\max} := \max C(p,k,M)$ of all odd p ≤ s, and let $h = (\varepsilon / C_{\max})^{1/2}$.
Replace h with ε, and deduce the upper bound of the weight

$$|w_{p,i}| = \frac{\binom{p}{i}}{|\sigma^{(p)}(0)|}\, h^{-p} = \frac{\binom{p}{i}}{|\sigma^{(p)}(0)|} \left(\frac{C_{\max}}{\varepsilon}\right)^{p/2}.$$

The term-by-term upper bound is

$$\max_{i} \binom{p}{i} \;\le\; \frac{2^{p}}{\sqrt{p+1}} \;\lesssim\; 2^{p}, \qquad \frac{1}{|\sigma^{(p)}(0)|} \;\le\; C_{p}(\text{value can be absorbing to a constant that only depends on } p),$$

and by the combined estimation of the consistent upper bound of [-M,M] and the k-order derivative, we get

$$C_{\max} \;\le\; \left(2(s+2)\sqrt{2M}\right)^{\,s+3}.$$

Merge and take the worst p=s for all odd p ≤ s, and get (package the mild and constant into a large O)

$$\max_{p\le s,i} |w_{p,i}| = O\Big(\varepsilon^{-s/2} \left(2(s+2)\sqrt{2M}\right)^{\,s(s+3)}\Big).$$

What the lemma 3.2 claims:

Pick an interval [-M,M]. For any target accuracy ε > 0 and any odd integer s, there is a one-hidden-layer neural network with the tanh activation that produces s outputs. Its p-th output approximates the power function y ↦ yˆp for every p = 0,1,..., s. All these approximations are uniformly accurate (the error is at most ε everywhere on the interval), and this remains true if we also look at the first k derivatives. The network needs only 3(s+1)/2 hidden neurons, and the lemma also gives an explicit (though large) upper bound on how big the weights might have to be.

Why this is plausible:

- The activation tanh is an odd function. With suitable scalings and shifts, sums of tanh can mimic odd powers yˆ(2n+1) very well (this is established in Lemma 3.1).
- Once we can mimic odd powers, there is a simple algebraic identity that recovers an even power from two nearby odd powers:

$$\frac{(y+\alpha)^{2n+1}-(y-\alpha)^{2n+1}}{2\alpha(2n+1)} = y^{2n} + (\text{lower even powers}).$$

This means that if we already approximated the lower even powers, we can use the difference on the left to get the next even power. Repeating this gives all even powers up to yˆs.

How the network is actually built:

1. Build a small set of "odd-power approximators" using tanh (from Lemma 3.1).
2. Evaluate those same approximators at three shifted inputs y - α, y, and y + α.
3. Combine these three stacks linearly, using the identity above, to solve for each even power.

How the error is kept in control:

Define E_p as the error for y^p.

- For odd p, Lemma 3.1 already gives $E_p \le \varepsilon$.
- For even p, the identity shows E_{2n} is a combination of E_{2n+1} (an odd power) and earlier E_{2k} with some binomial coefficients. A simple induction, plus choosing a small shift $\alpha \approx 1/s$, keeps these errors from blowing up. At the end we slightly tighten the target used in step1 so that all $E_p \le \varepsilon$.

What the weight bound tells us:

To reach very small error ε for many powers (large s), some coefficients in the linear combination must be large (the bound in the lemma quantifies this growth). Practically, it says the method works in principle but the network can become numerically heavy as you push s up or ε down.