1.

Lemma 3.1

Statement:
For any integer s (the degree of a polynomial you want), you can construct a shallow neural network with tanh activation that approximates the function y^s (a monomial) to any desired accuracy. Moreover, the network's size does not blow up uncontrollably—it can be kept under control.

Idea:

- Think of tanh(x), the hyperbolic tangent. Its derivatives at 0 produce values that can be combined to mimic powers of x.
- By carefully choosing weights (the multipliers in the network), you can make a neuron output something close to y^s.
- This lemma proves: *Yes, a single-hidden-layer neural network can approximate y^s as closely as we want.*

Analogy:
It's like using LEGO bricks (tanh functions) to build a model of a curve y^s. With enough bricks, you can get arbitrarily close to the real shape.

Lemma 3.2

Statement:
Not only can we approximate a single monomial y^s, but we can also control the accuracy and the size of the network uniformly across all powers $p \leq s$. In other words, the same type of network construction works for all lower powers too, not just the highest one.

Idea:

- Lemma 3.1 gave us y^s.

- Lemma 3.2 strengthens this: we can approximate the entire family $\{y, y^2, \ldots, y^s\}$ using shallow networks.
- This is important because real-world functions are usually combinations of many powers of y. So, if we want to approximate something like $3y^2 - 5y^4$, we need approximations for multiple powers at once.

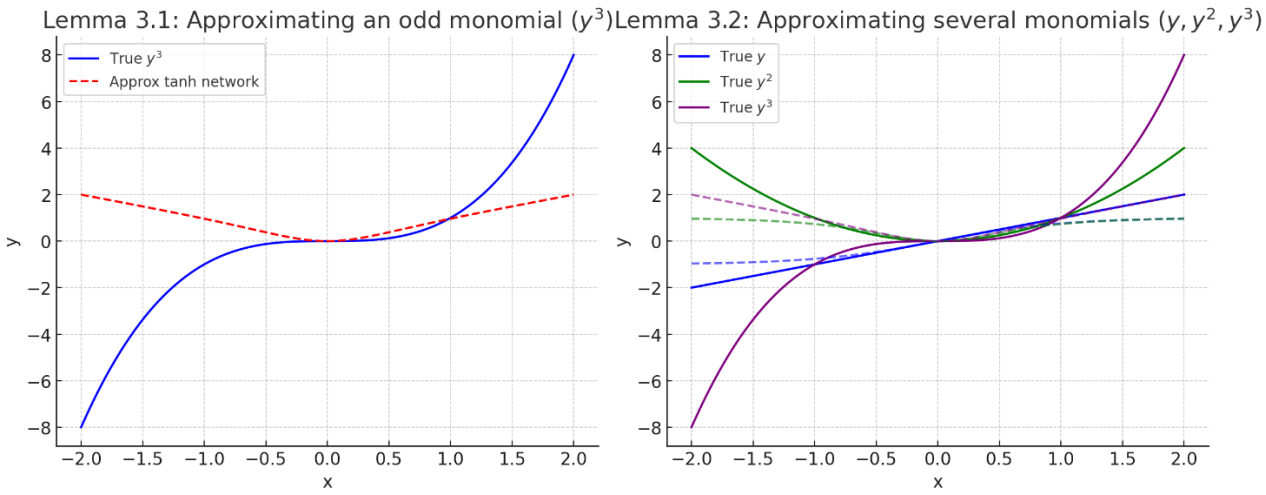
Why it matters:
This lemma is the bridge to approximating general polynomials. Since polynomials can approximate continuous functions (Weierstrass theorem), and neural networks can approximate polynomials, it follows that neural networks can approximate any continuous function.

Intuitive Example

Suppose you want a network to approximate $f(y) = y^3$.

- Lemma 3.1 says: *Yes, you can make a tanh neural network that outputs something arbitrarily close to y^3.*
- Lemma 3.2 says: *If you also wanted y^2 and y^3 at the same time, the construction still works. In fact, you can get all powers up to some degree s consistently.*

Imagine drawing y^3 on a graph. The network learns to bend the smooth tanh curve so that, when combined properly, it matches the cubic curve. Lemma 3.2 ensures this trick works for multiple curves simultaneously.



Lemma 3.1 (left): A tanh-based shallow network can approximate an odd monomial like y^3.

Lemma 3.2 (right): The same construction extends to several monomials up to degree s (here y,y^2,y^3), showing that the network can approximate them all simultaneously.

2.

How should one choose the finite-difference step h to balance truncation error $O(h^k)$ and numeric round-off / noise amplification in practice?

The error bounds shown are $O(h^2)$ etc.; what constants (dependent on σ and domain of x) control those bounds? How do they scale with p and |x|?