

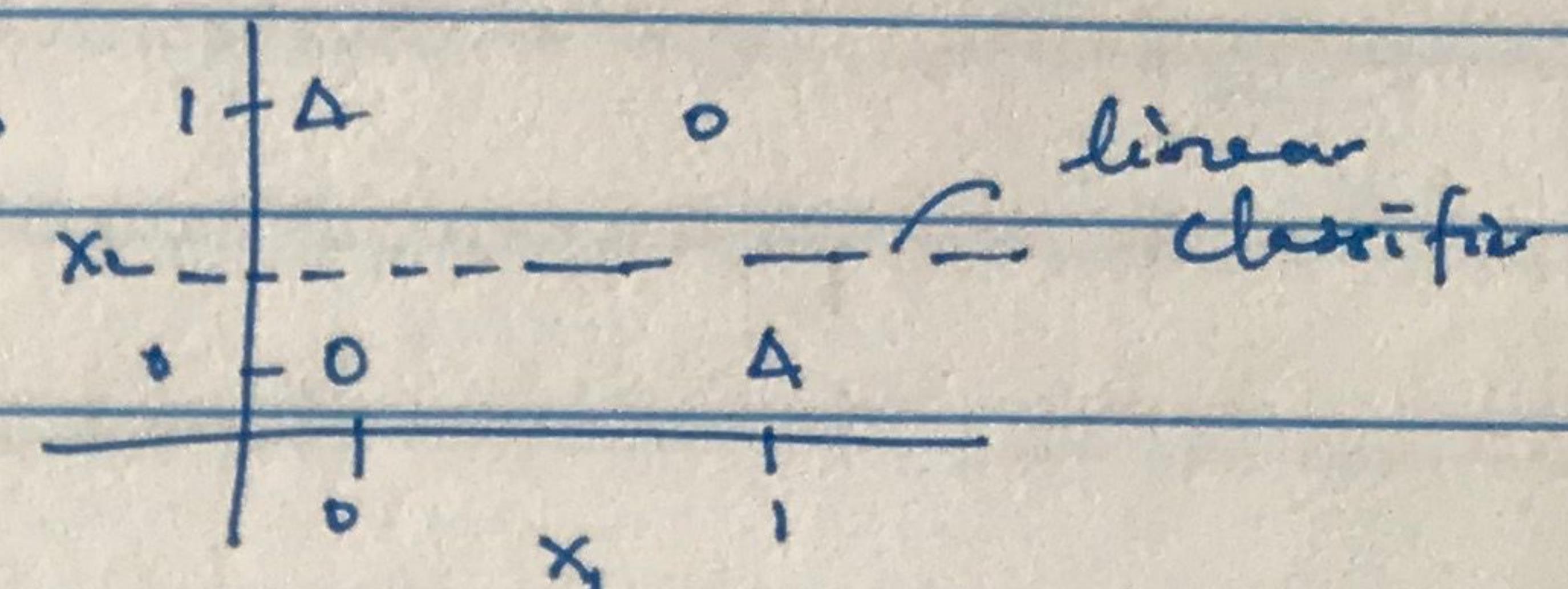
DEEP FEED FORWARD NETWORKS

* Why do we need non-linearity?

⇒ Linear classifiers cannot separate non-linearly separable data.

For example, consider XOR.

- A linear classifier $y = w^T x + b \Rightarrow$ cannot achieve '0' error.

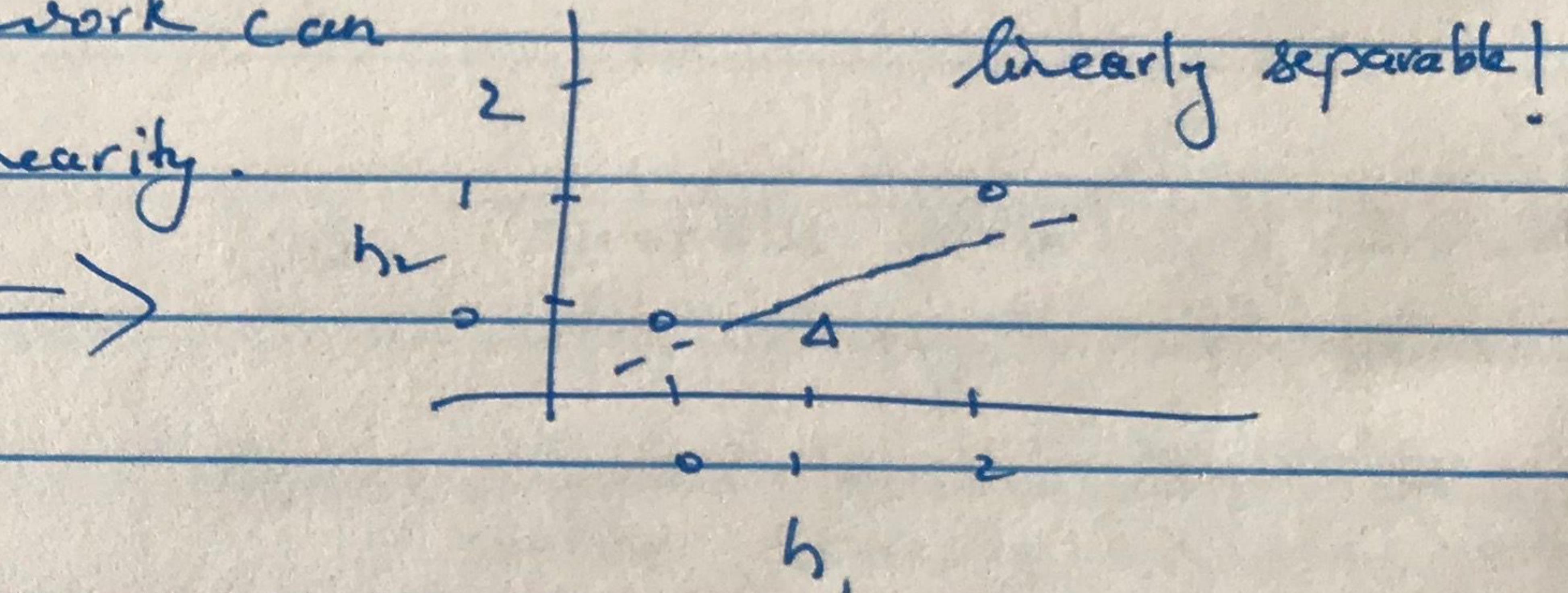


- A single hidden layer neural network can achieve '0' error with non-linearity.

$$y = f(w_2^T h + b_2)$$

$$h = g(w_1^T x + b_1)$$

non-linearity



* How to construct a deep feedforward network?

1. Cost Function

Principle of maximum likelihood states that if $p(y|x, \theta)$ is given then cost $f^n = -\log p(y|x, \theta)$

More formally, for $p_{model}(y|x)$

$$J(\theta) = - \mathbb{E}_{x,y \sim P_{data}} \log p_{model}(y|x)$$

Thus we use cross entropy loss function.

+ Cross entropy loss can adapt to the choice of output unit.

2. Output unit

- Linear output unit

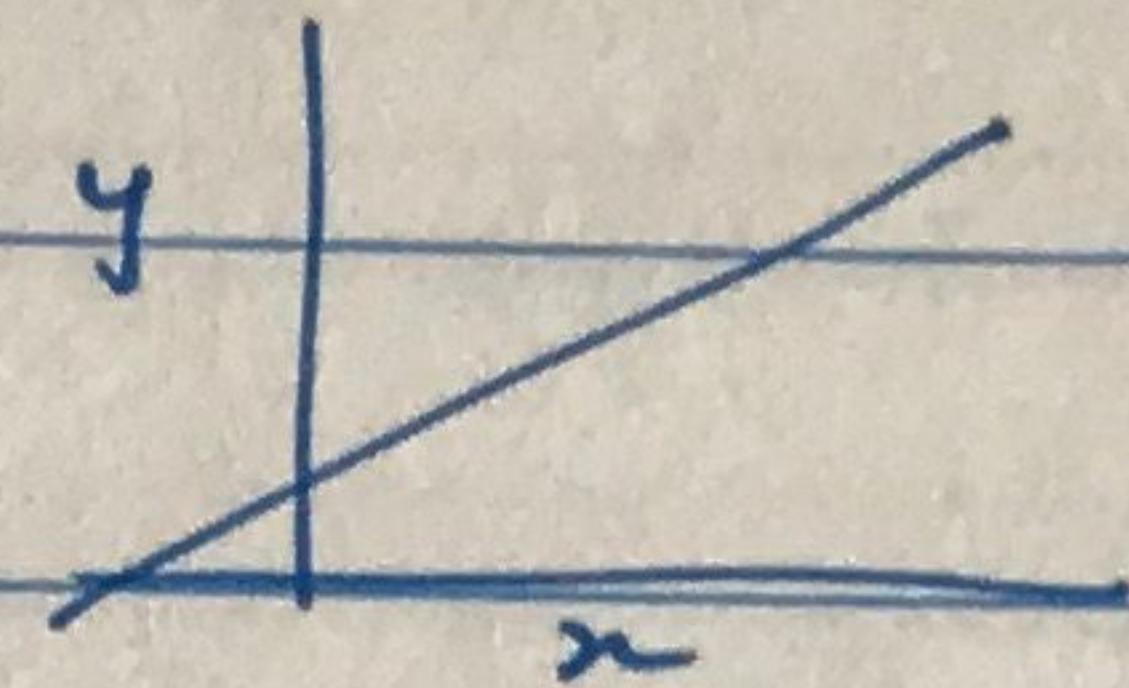
$$\hat{y} = \omega^T h + b$$

$$\rightarrow p(y|x) = N(y; \hat{y}, I)$$

Thus it follows gaussian output distribution

\Rightarrow Cross entropy loss will become mean square error loss.

$$J(\theta) = \frac{1}{2} \mathbb{E}_{x,y \sim \hat{P}_{\text{data}}} \|y - \hat{y}\|^2 + \text{const.}$$



- Sigmoid unit

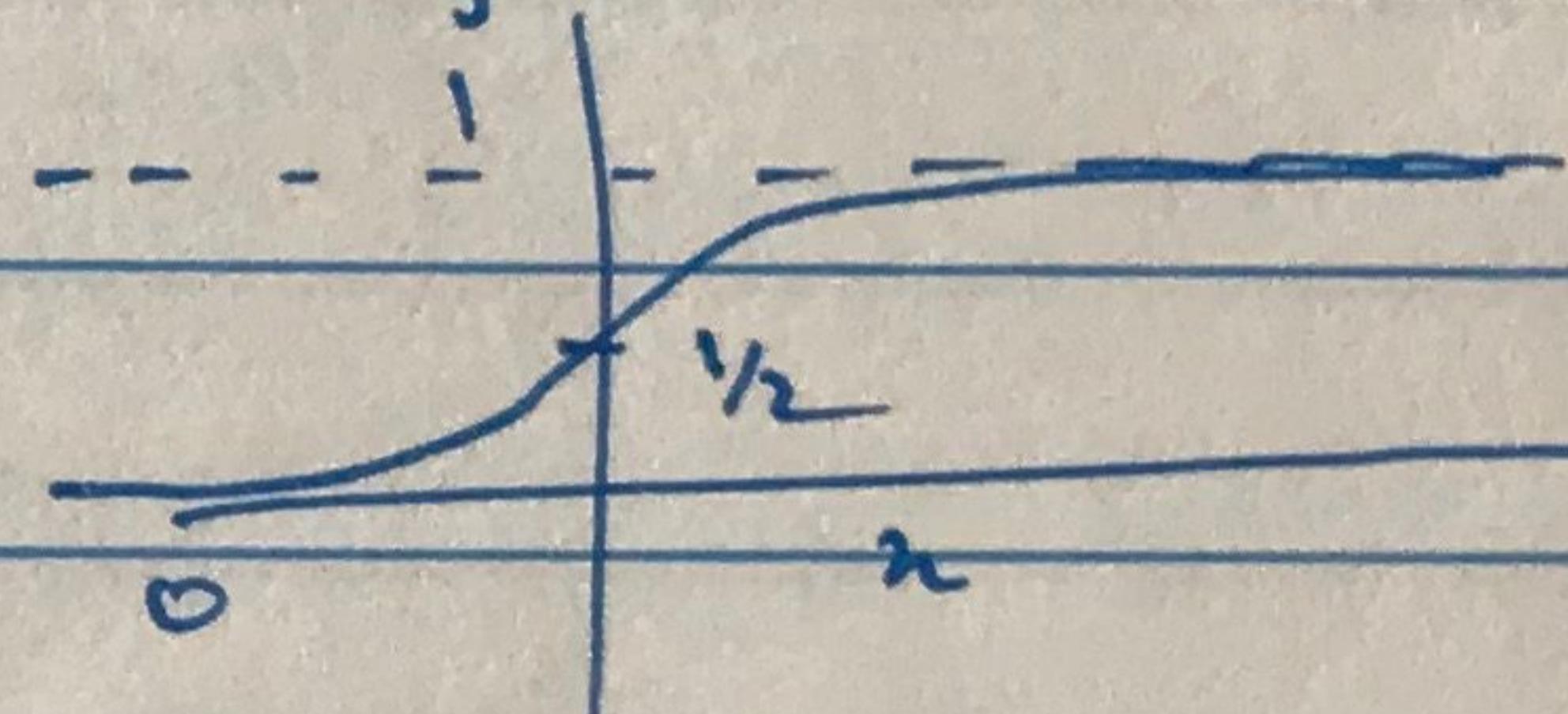
$$\hat{y} = \sigma(\omega^T h + b),$$

$$\text{where } \sigma(z) = \frac{1}{1+e^{-z}}$$

Sigmoid units generate $\{0, 1\}$ probability distribution & hence follow bernoulli distribution.

Pros: Generates probability distribution for binary classification

Cons: Saturates at higher/lower values, causing vanishing gradient



Assume $\log \tilde{p}(y) = y_2$, where $\tilde{p}(.)$ is un-normalized prob. distribution

$$\Rightarrow \tilde{p}(y) = \exp(y_2)$$

$$\therefore p(y) = \frac{\exp(y_2)}{\sum_{y=0}^1 \exp(y_2)}$$

$$= \sigma((2y-1)^2)$$

$$p(0) = \sigma(-2) = \frac{1}{1+e^{-2}} = \frac{e^{-2}}{1+e^{-2}}$$

$$p(1) = \sigma(2) = \frac{1}{1+e^{-2}}$$

$$p(0) + p(1) = \frac{1}{1+e^{-2}} + \frac{e^{-2}}{1+e^{-2}} = 1$$

(Cross entropy loss - $\log p(y|x)$)

cancels the $\exp()$ effect of σ .

So the output does not saturate.

- Softmax unit

Softmax follows multinoulli distribution.

$$Z = \omega^T b + b$$

$$z_i = \log \hat{p}(y=i|x)$$

$$\text{Softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

$$\sum_j \exp(z_j)$$

Pros Generates probability distribution for multi-class classification

Cons Saturates for large values.

* Cross entropy simulates log softmax

$$\therefore \text{log Softmax} = z_i - \log \sum_j \exp(z_j)$$

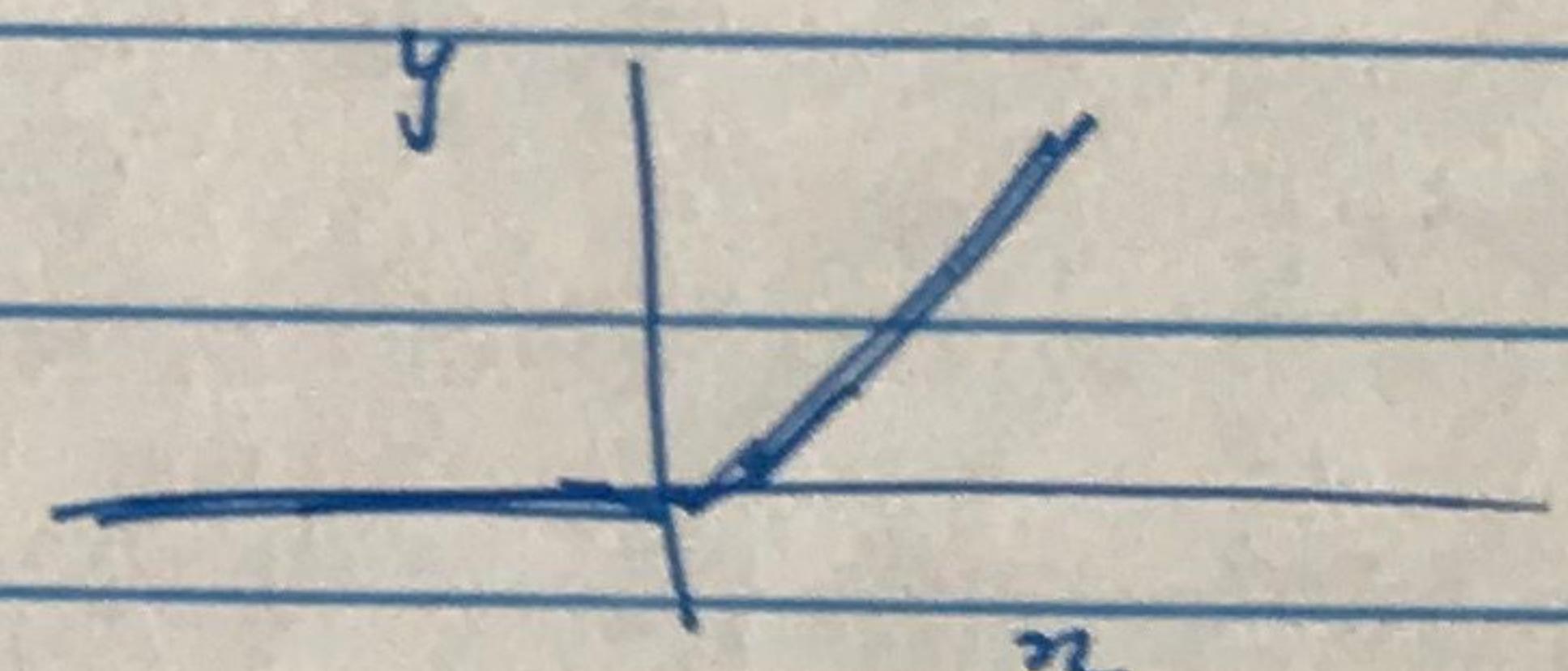
$$\approx z_i - \max_j z_j$$

Thus log softmax penalizes if the most probable class $z_i \neq \max z_j$

3. Hidden unit

- ReLU (Rectifier Linear Unit)

$$g(z) = \max(0, z)$$

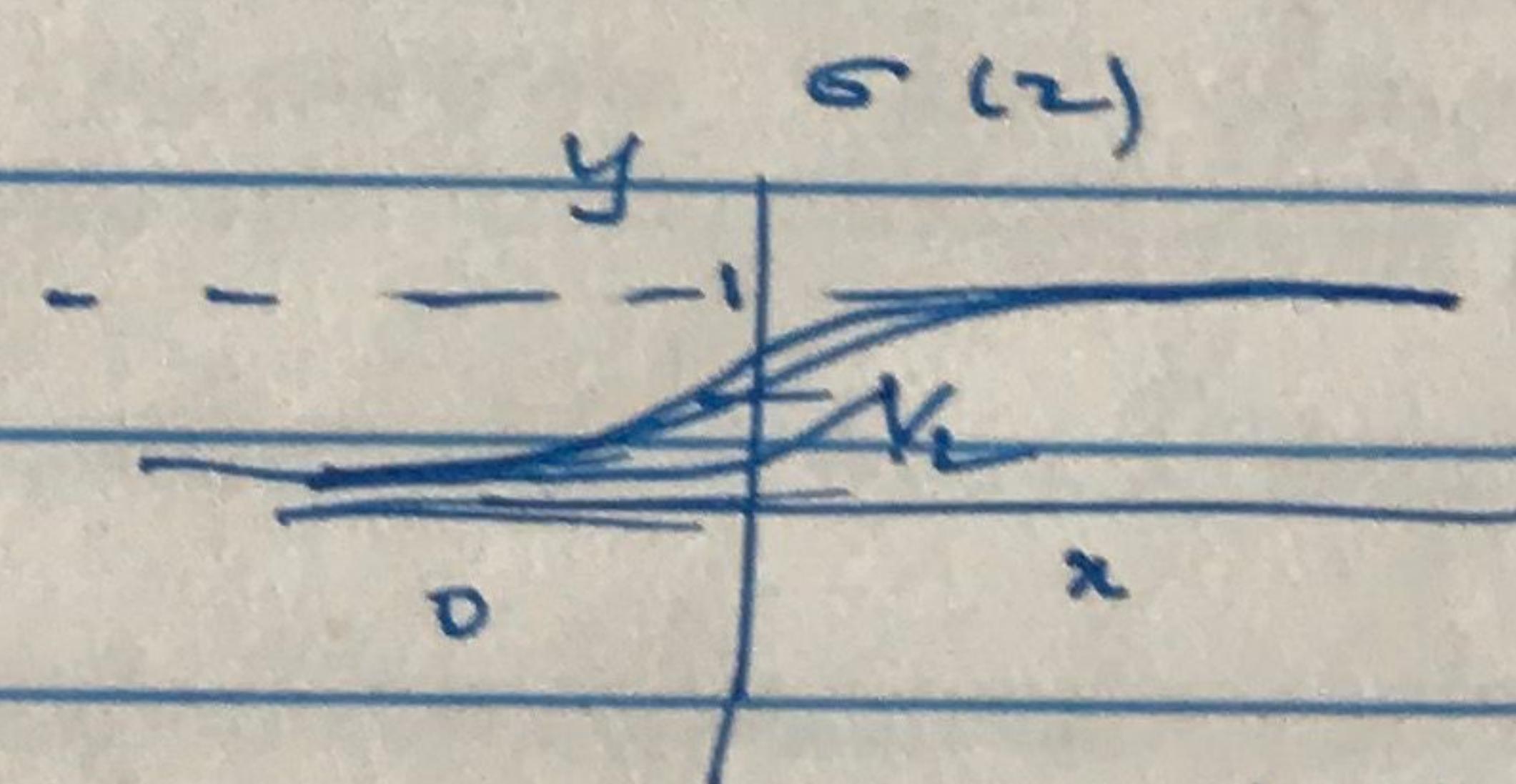


ReLU does not saturate for any value and hence good choice for hidden unit.

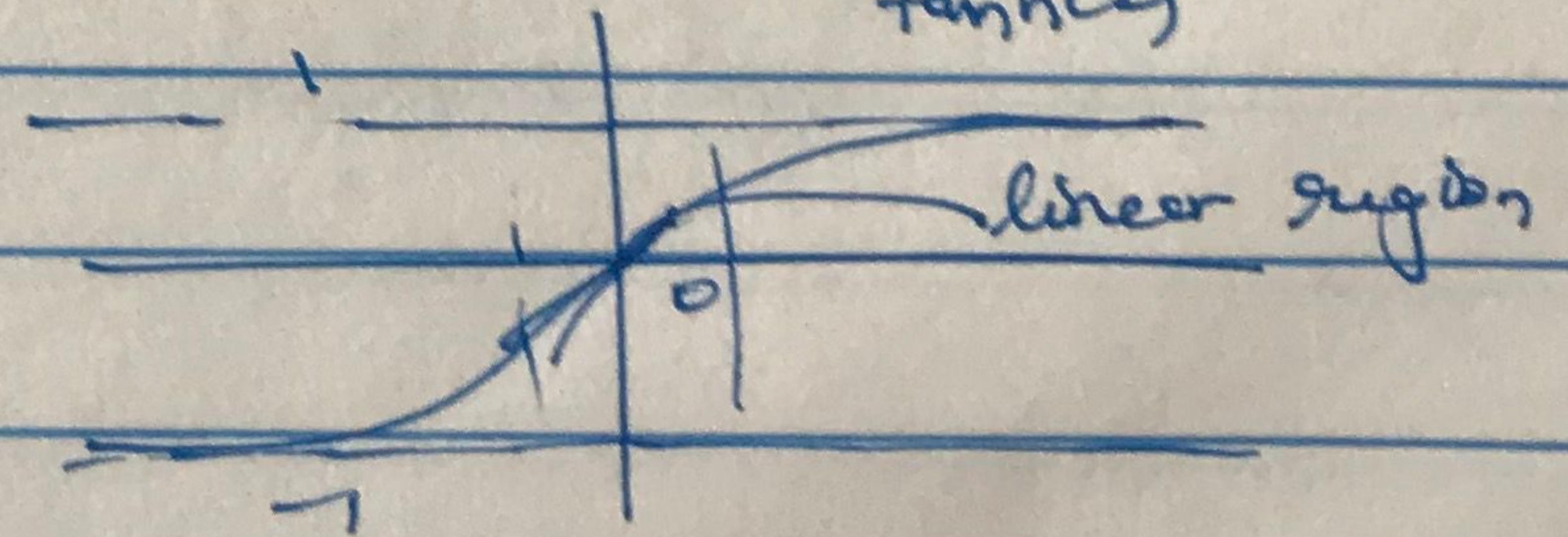
- Logistic Sigmoid and

- Hyperbolic Tangent

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



$$\tanh(z) = 2\sigma(2z) - 1$$



both functions saturate for large values and hence gradient is small.

But tanh is preferred over sigmoid since it has linear region near '0' and hence it behaves well.

+ Used when probability distribution is required.

4. Architecture Considerations

- Number of layers
- Number of hidden units per layer
- Connectivity between layers
 - dense (MLP)
 - sparse (CNN)

Universal Approximation theorem states that a feedforward neural network with at least one hidden layer with 'squashing' activation function can approximate any functionality.

- However one hidden layer would require $O(2^n)$ degrees of freedom to simulate any 2^2^n binary function. Thus it would require exponential number of neurons in one hidden layer.
- Thus having more hidden layer reduces the requirement of exponential number of neurons per layer.
- Main theorem of Montufar et al. states that number of regions carved out by a deep rectifier network with depth l , d inputs, and n units per hidden layer is
$$O\left(\binom{d}{d}^{d(l-1)} n^d\right)$$
- Having larger depth reduces the no. of hidden units and still achieves any functionality.