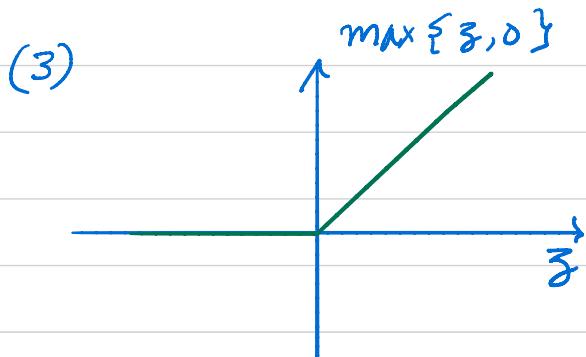
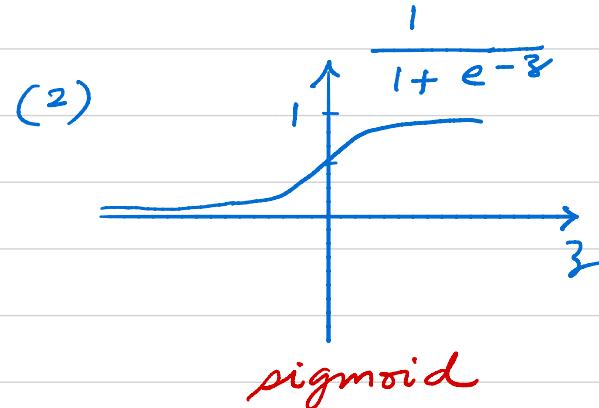
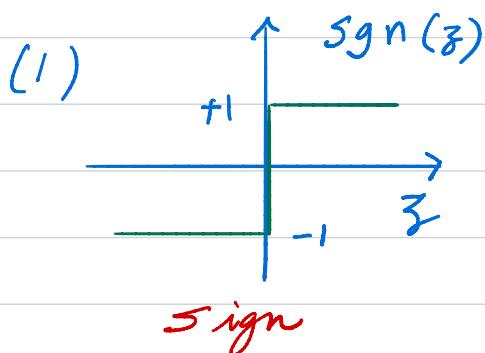


Application: Neural Networks

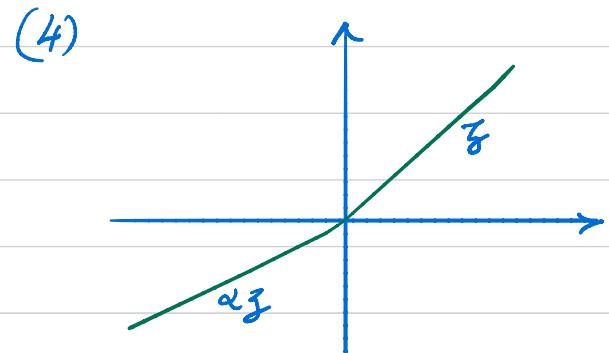
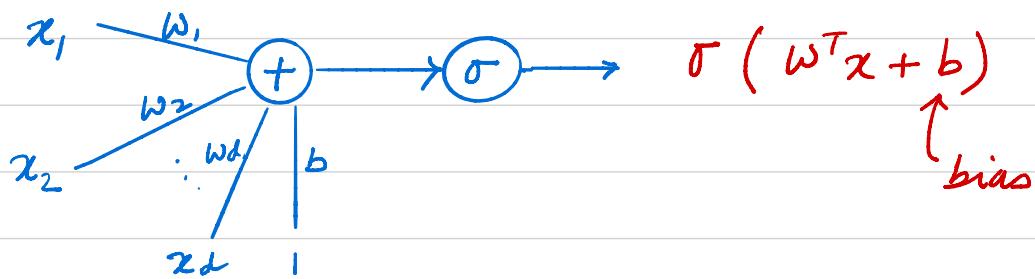
- Neuron is a non-linear function, which takes  $z \in \mathbb{R}$  as input and produces  $\sigma(z) \in \mathbb{R}$  as output

$$z \rightarrow \sigma \rightarrow \sigma(z)$$

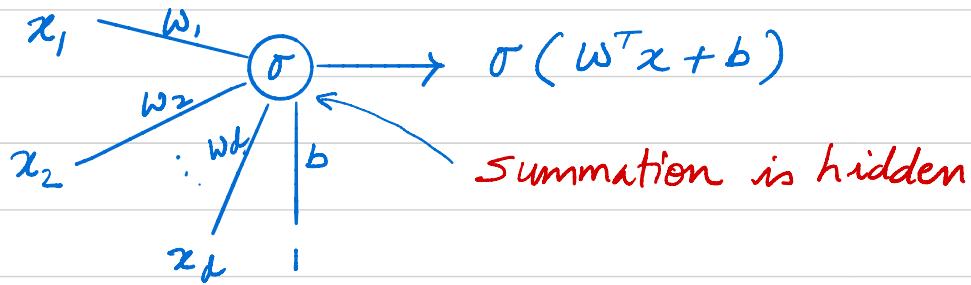
Examples of  $\sigma$ :



Rectifier Linear Unit  
(ReLU)

Vector Input

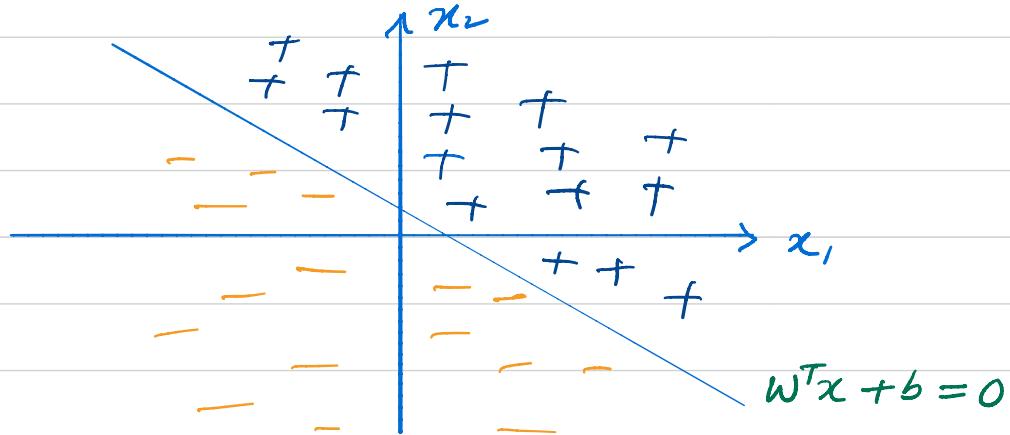
## Simplified Representation



- bias  $b$  is important, e.g., if  $\sigma$  is the sgn function, we can output  $+1$  if  $w^T x \geq -b$   
 $-1$  if  $w^T x < -b$  instead of  $w^T x \geq 0$ .

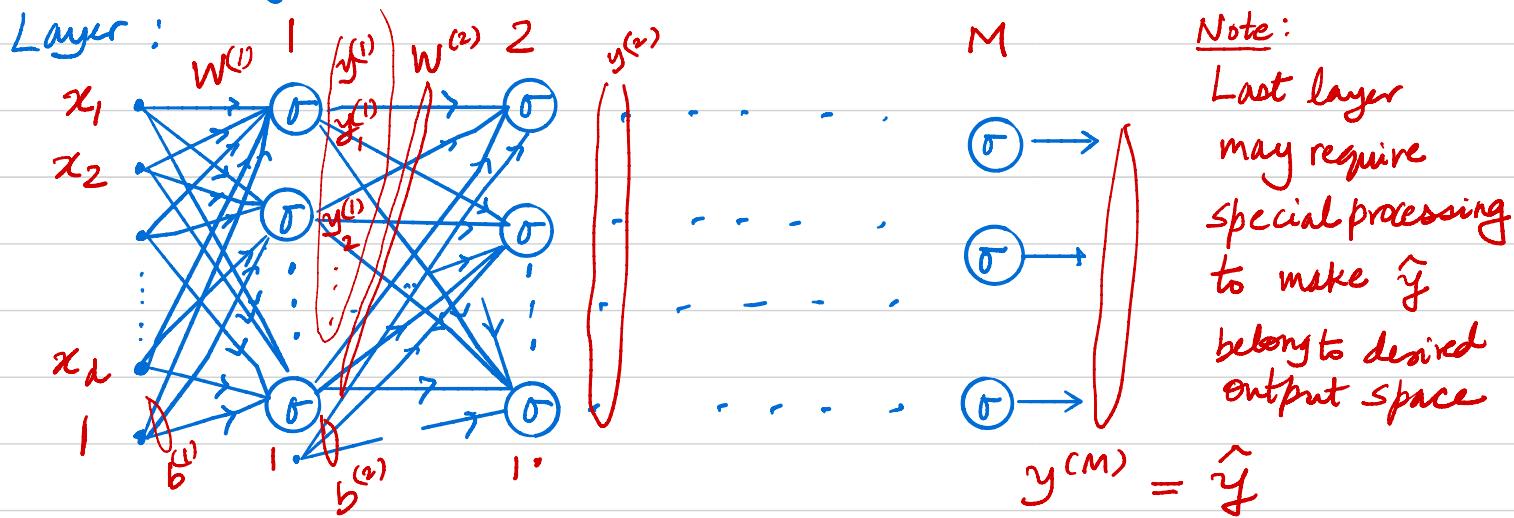
$w^T x + b = 0$  is a hyperplane in  $\mathbb{R}^d$ .

and neuron simply divides the input space into two parts corresponding to  $+1$  and  $-1$



Multilayer NN's allow us to divide high dimensional space in more complicated ways and approximate functions mapping inputs and outputs of NN

# Multi-layer Neural Network (Feed Forward)



- number of neurons in each layer can be different
- All weights on edges connecting layers  $m-1$  and  $m$  are in matrix  $W^{(m)}$ , with  $W_{ij}^{(m)}$  being the weight connecting output  $j$  of layer  $(m-1)$  with neuron  $i$  of layer  $m$
- Input to network is vector  $x$ ; output of layer  $m$  is vector  $y^{(m)}$

$$y_i^{(1)} = \sigma(x_i^{(1)}), \text{ with } x_i^{(1)} = \sum_j W_{ij}^{(1)} x_j + b_i^{(1)}$$

This can be written compactly as :

$$y^{(1)} = \sigma(x^{(1)}) \text{ with } x^{(1)} = W^{(1)}x + b^{(1)}$$

$\uparrow$

$\sigma$  applied to each element of  $x^{(1)}$

$$y^{(1)} = \sigma(x^{(0)}) \text{ , with } x^{(0)} = W^{(0)}x + b^{(0)}$$

Similarly,

$$y^{(2)} = \sigma(x^{(2)}) \text{ , with } x^{(2)} = W^{(2)}y^{(1)} + b^{(2)}$$

:

$$y^{(M)} = \sigma(x^{(M)}) \text{ , with } x^{(M)} = W^{(M)}y^{(M-1)} + b^{(M)}$$

Goal is choose the weights  $W^{(1)}, \dots, W^{(M)}, b^{(1)}, \dots, b^{(M)}$   
so that the output of last layer, a.k.a. output  
of NN approximates some desired function  
well, i.e., we want

$$\hat{y} = y^{(M)} \approx f^*(x) = y$$

↑ unknown

possibly multidimensional

How is the function approximated? Use labelled  
training data, i.e. inputs for which we know  
correct output, even though  $f^*$  is unknown:

$$(x[1], y[1]), (x[2], y[2]), \dots, (x[N], y[N])$$

Minimize "empirical" loss on training data

$$J = \sum_{n=1}^N L(y[n], \hat{y}[n])$$

Where  $\hat{y}[n]$  is output of NN when input  
is  $x[n]$ .

- In Case the desired output  $y$  is (cont.) scalar, i.e.  $f: \mathbb{R}^d \rightarrow \mathbb{R}$ , then  $L$  is often taken to be

The Square loss :  $L(y, \hat{y}) = (y - \hat{y})^2$

$\hat{y} = y^{(M)}$  means last layer has only one neuron

- $J = \sum_{n=1}^N L(y[n], \hat{y}[n])$

$\uparrow$  assume cont. differentiable

$$= y^{(M)}[n]$$

- Note that  $J$  is a function of  $w^{(1)}, \dots, w^{(M)}, b^{(1)}, \dots, b^{(M)}$
- We wish to minimize  $J$  using a gradient descent procedure.

- Need to compute gradient of  $J$  w.r.t. all weights
- Since  $J$  is a sum, we can compute the gradient for each term separately.  
we drop "[n]" for convenience.
- To compute gradient we need :

$$\frac{\partial L}{\partial w_{ij}^{(l)}} \text{ for each } l, i, j.$$

and  $\frac{\partial L}{\partial b_i^{(l)}}$  for each  $l, i$ .

## Back Propagation Algorithm

- Exploit network structure and chain Rule to compute gradient terms efficiently

Layer M Recall  $y^{(M)} = \sigma(x^{(M)})$ ,  $x^{(M)} = W^{(M)}y^{(M-1)} + b^{(M)}$

$$\frac{\partial L}{\partial W_{ij}^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} \cdot \frac{\partial y_i^{(M)}}{\partial W_{ij}^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} \frac{\partial y_i^{(M)}}{\partial x_i^{(M)}} \frac{\partial x_i^{(M)}}{\partial W_{ij}^{(M)}}$$

$\frac{\partial L}{\partial y_i^{(M)}}$  can easily be computed in practice for each data point  $(x^{[n]}, y^{[n]})$ .

For example if  $y^{(M)}$ ,  $y$  are scalar, and  $L(y, y^{(M)}) = (y - y^{(M)})^2$ . Then, for  $(x^{[n]}, y^{[n]})$ ,

$$\frac{\partial L}{\partial y^{(M)}}(y^{[n]}, y^{(M)[n]}) = 2(y^{(M)[n]} - y^{[n]}).$$

$$\frac{\partial y_i^{(M)}}{\partial x_i^{(M)}} = \frac{d \sigma(x_i^{(M)})}{d x_i^{(M)}} = \sigma'(x_i^{(M)})$$

(assuming  $\sigma$  cont. diff.)

$$x_i^{(M)} = \sum_k W_{ik}^{(M)} y_k^{(M-1)} + b_i^{(M)}$$

$$\Rightarrow \frac{\partial x_i^{(M)}}{\partial W_{ij}^{(M)}} = y_j^{(M-1)}$$

Thus,

$$\frac{\partial L}{\partial W_{ij}^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} \cdot \sigma'(x_i^{(M)}) y_j^{(M-1)}$$

$$\text{Similarly, } \frac{\partial L}{\partial b_i^{(m)}} = \underbrace{\frac{\partial L}{\partial y_i^{(m)}}}_{=1} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial b_i^{(m)}}$$

$$= \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)})$$

Layer m,  $1 \leq m \leq M$

Recall that  $y^{(m)} = \sigma(x^{(m)})$  with  $x^{(m)} = W^{(m)}y^{(m-1)} + b^{(m)}$

$$\frac{\partial L}{\partial W_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \underbrace{\frac{\partial y_i^{(m)}}{\partial x_i^{(m)}}}_{\sigma'(x_i^{(m)})} \cdot \underbrace{\frac{\partial x_i^{(m)}}{\partial W_{ij}^{(m)}}}_{y_j^{(m-1)}}$$

just as in layer M

Note that  $y^{(m)}$  is related to  $x^{(m+1)} = W^{(m+1)}y^{(m)} + b^{(m+1)}$

$$\begin{aligned} \Rightarrow \frac{\partial L}{\partial y_i^{(m)}} &= \sum_k \frac{\partial L}{\partial x_k^{(m+1)}} \frac{\partial x_k^{(m+1)}}{\partial y_i^{(m)}} \\ &= \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} \cdot \underbrace{\frac{\partial y_k^{(m+1)}}{\partial x_k^{(m+1)}}}_{\sigma'(x_k^{(m+1)})} \cdot \underbrace{\frac{\partial x_k^{(m+1)}}{\partial y_i^{(m)}}}_{W_{ki}^{(m+1)}} \\ &\quad \xrightarrow{\text{computed in previous step}} \\ &= \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} \sigma'(x_k^{(m+1)}) W_{ki}^{(m+1)} \end{aligned}$$

Similarly,

$$\frac{\partial L}{\partial b_i^{(m)}} = \underbrace{\frac{\partial L}{\partial y_i^{(m)}}}_{\text{above}} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial b_i^{(m)}}$$

$$\sigma'(x_i^{(m)}) \quad \boxed{1}$$

## Back Propagation Summary

(1) Compute  $\frac{\partial L}{\partial y_i^{(m)}}$  for all  $i$

(2) For  $m = M$  down to 1.

$$\frac{\partial L}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \sigma'(x_i^{(m)}) y_j^{(m-1)}$$

$$\frac{\partial L}{\partial b_i^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \sigma'(x_i^{(m)})$$

Where  $\frac{\partial L}{\partial y_i^{(m)}} = \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} \sigma'(x_k^{(m+1)}) w_{ki}^{(m+1)}$

### Remarks

1. Gradient computation can be performed in parallel for different data points

2. In practice iterates are generated using gradients of only subset of data points for each iteration:

(Batch) Stochastic Gradient Descent (SGD)