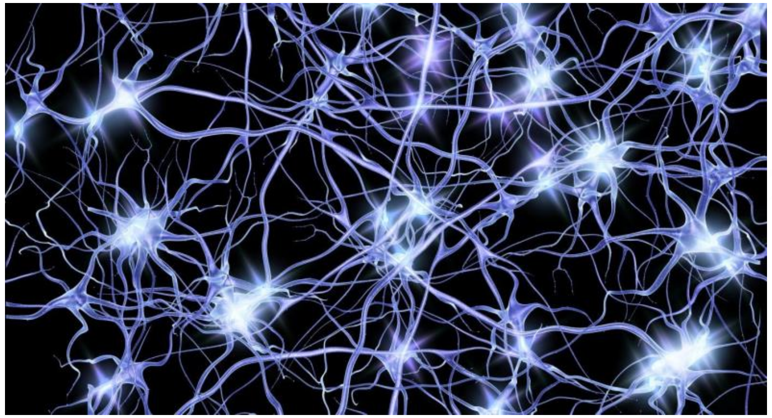
NEURAL NETWORK

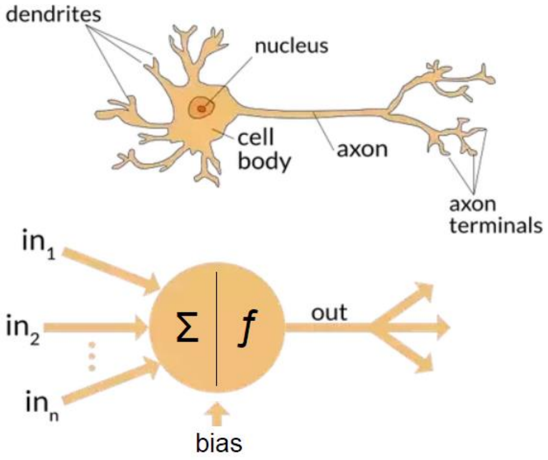
RESMI SURESH

ASSISTANT PROFESSOR, IIT GUWAHATI

NEURAL NETWORK STRUCTURES



- Neural networks are modeled based on the structure of human brain
  - Have the possibility of capturing the human cognitive processes in future
  - Basic notion
    - All cognitive processes occur in brain
    - If one were able to understand the structure of the brain and then map functions to these structures, then one could have an in-silico system that will be able to perform cognitive tasks
    - Brain structure
      - Most fundamental unit – neurons
      - Neurons are connected to each other so that information in terms of electrical and chemical signals can be exchanged for collective decision making
- These structures are quite useful in data science and ML even viewed purely as a nonlinear model form
- We would be discussing the engineering viewpoint
- Computational neural networks model the fundamental component and the interconnections through mathematical equations

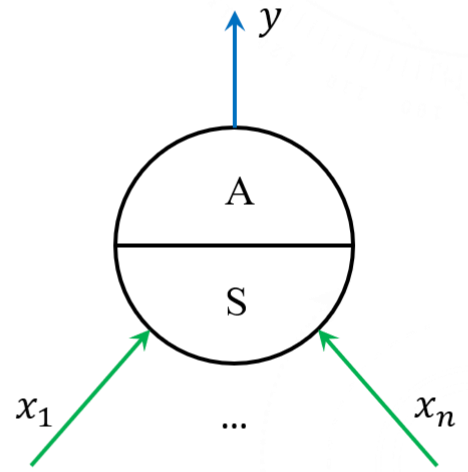


NEURAL NETWORK STRUCTURES

- A neuron in a neural network receives multiple inputs from other neurons and/or exogenous inputs $(x_1, \ldots, x_n)$ and generates an output (y)
- Each node/neuron is partitioned into two components/functions (S and A)
  - S acting on $(x_1, \ldots, x_n)$ to provide an intermediate output o
  - Activation function A acting on o to give the final output y
  - S is generally a summation function

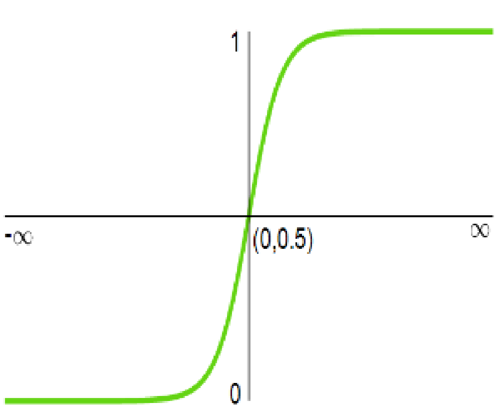
$$y = A(o);$$

$$o = S(x_1, x_2, \ldots, x_n) = \sum x_i$$



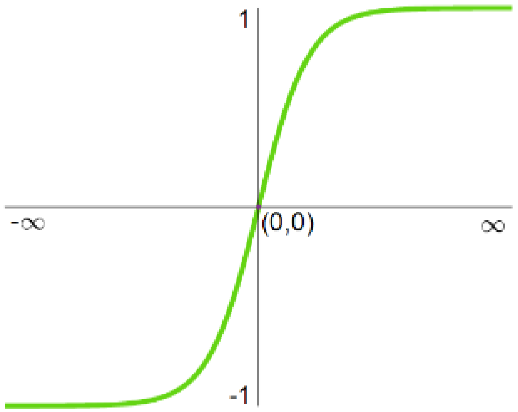
NEURAL NETWORK STRUCTURES
ACTIVATION FUNCTIONS

- Some of the popular activation functions used in neural networks



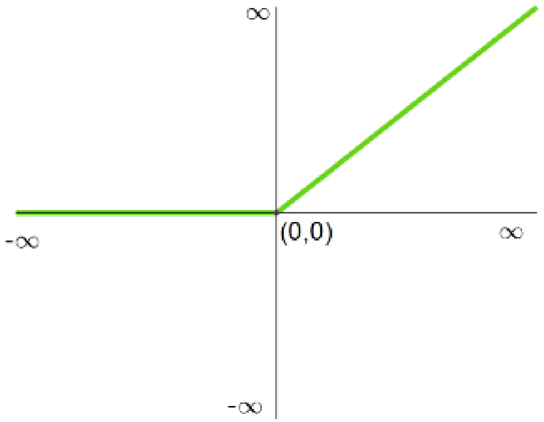
(a) Sigmoid function

$$A(o) = \frac{1}{1 + e^{-o}}$$



(b) tanh function

$$A(o) = \frac{e^{-o} - e^{o}}{e^{-o} + e^{o}}$$



(c) Relu function

$$A(o) = \begin{cases} o & \forall\ o \geq 0 \\ 0 & \forall\ o < 0 \end{cases}$$

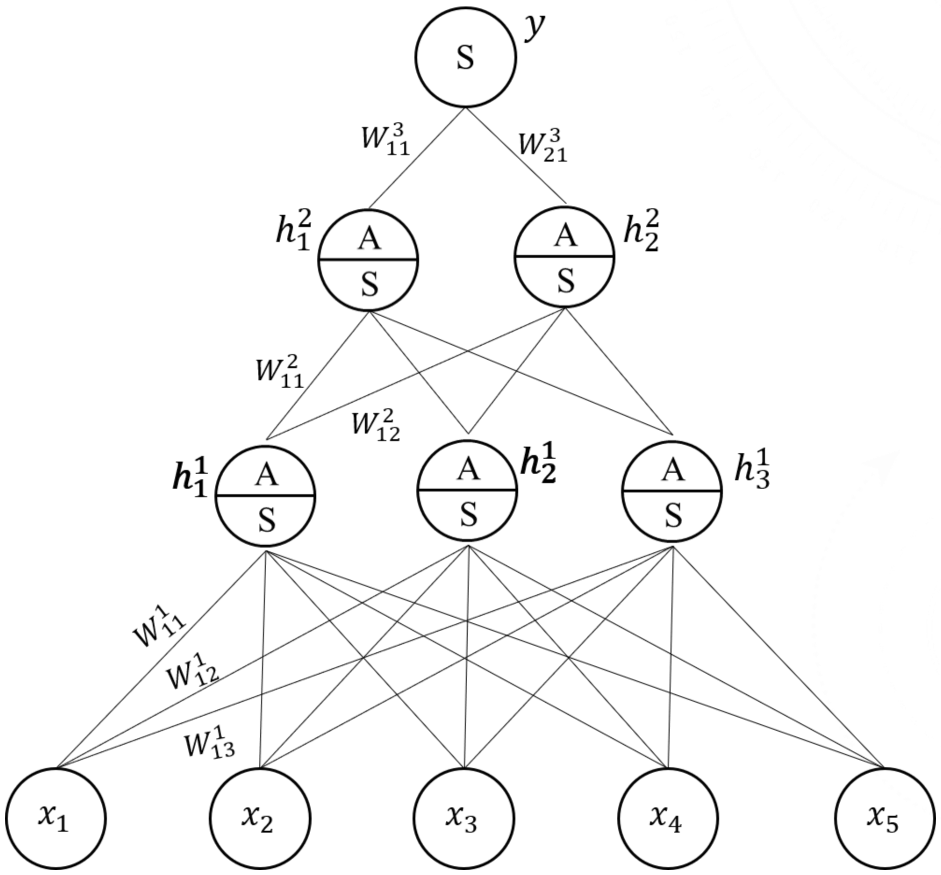
FULLY CONNECTED NEURAL NETWORK STRUCTURE

- Bottom most layer – input layer with input nodes (representing input variables)
    - Output of an input node is the value of the input variable
- Top most layer – output layer with output node (representing output variable)
- Layers in between – hidden layers
- Each node in a hidden layer will be connected to all the nodes in the previous layer
- The strength of each connection is represented by weights
- The output of j[th] node in the k[th] layer is calculated using

$$h_{k,j} = A(o_{k,j})$$

$$o_{k,j} = S\left(h_{k-1,1}, \dots, h_{k-1,r_{k-1}}\right) = \sum_{i=1}^{l=r_{k-1}} w_{k,lj}\, h_{k-1,l}$$

where r_{k-1} is the number of nodes in the (k-1)[th] layer



FULLY CONNECTED NEURAL NETWORK STRUCTURE

- If we have an m layer network (excluding input layer) and a single output y, then the predicted value of output from the given neural network is

$$\hat{y} = h_{m,1}$$

- Given a neural network structure, all the weights (W) and values of input variables, output can be calculated – forward calculation or propagation
  - Example: If sigmoid activation function is used and the inputs are $x_1 = -3.2$, $x_2 = 4.1$ and $x_3 = 0.163$,

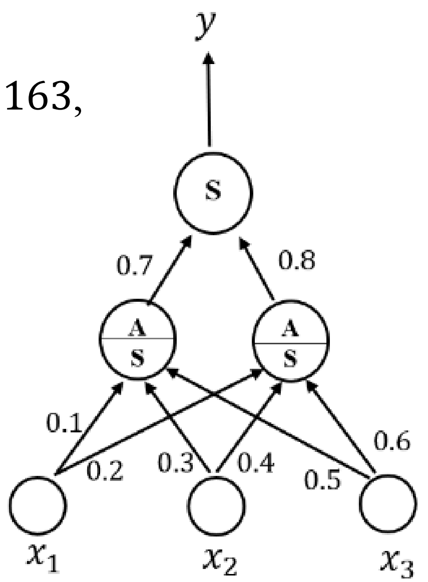
$$O_{1,1} = 0.1x_1 + 0.3x_2 + 0.5x_3 = 0.9915$$

$$O_{1,2} = 0.2x_1 + 0.4x_2 + 0.6x_3 = 1.0978$$

$$h_{1,1} = A\left(O_{1,1}\right) == \frac{1}{1 + e^{-1.0015}} = 0.7294$$

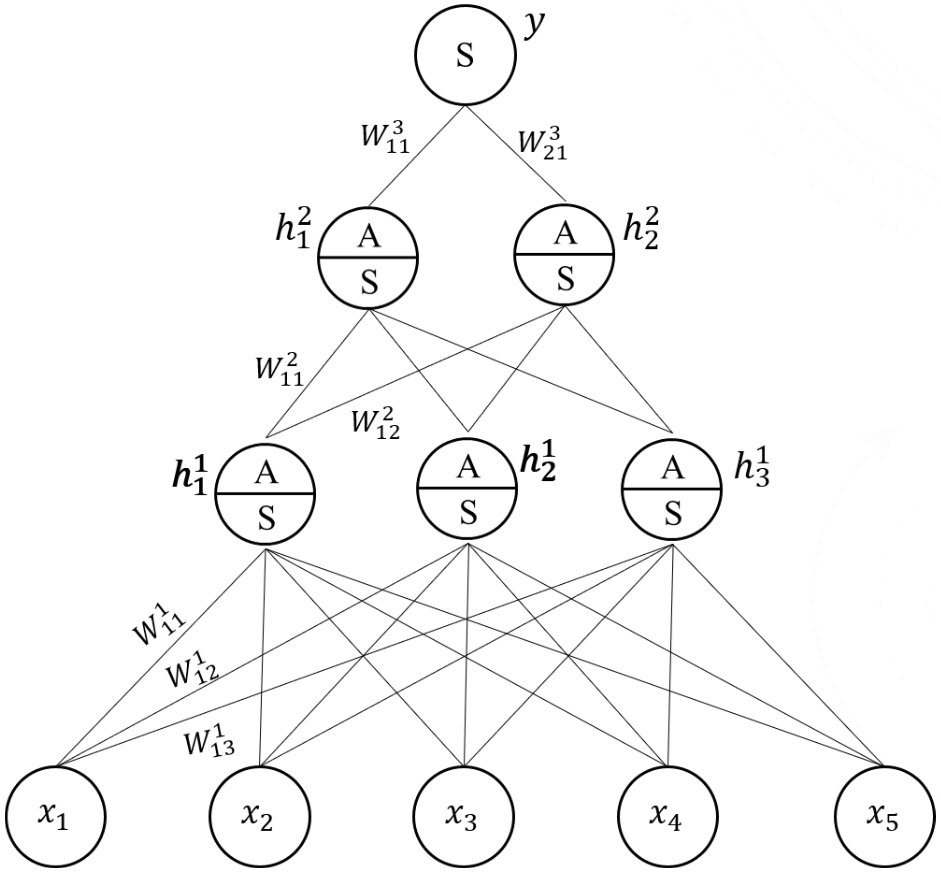
$$h_{1,2} = A\left(O_{1,2}\right) == \frac{1}{1 + e^{-1.1178}} = 0.7498$$

$$y = 0.7h_{1,1} + 0.8h_{1,2} = 1.1104$$



FULLY CONNECTED NEURAL NETWORK STRUCTURE

- Can be extended to multiple outputs
- Neural network is a non-linear model for the function approximation problem
- Though an explicit model form can be written, it is more convenient to refer to the model as a weights structure, W
- Use of a NN require
    - A choice for the structure
        - No. of hidden layers, choice of activation function and no of nodes in each hidden layer
    - Choice of weights given structure
- How do we decide the neural network structure and weights?



TRAINING OF NEURAL NETWORKS

- The neural network structure is usually decided based on prior knowledge or by trial-and-error
- Training a network implies the evaluation of optimal weights for the given network structure for the data available
- Optimization problem

$$\underset{W}{Min} \sum_{s=1}^{m} (\hat{y}^s(W) - y^s)^2$$

  - s – sample number; W – weight matrix (decision variable)
  - Predicted output will be a complex function of W
  - Objective – minimization of prediction error
  - Can we solve this using any of the methods we learned in optimization?
    - Non-linear programming problem
    - Any solution strategy to solve unconstrained NLP like steepest descent algorithm can be used

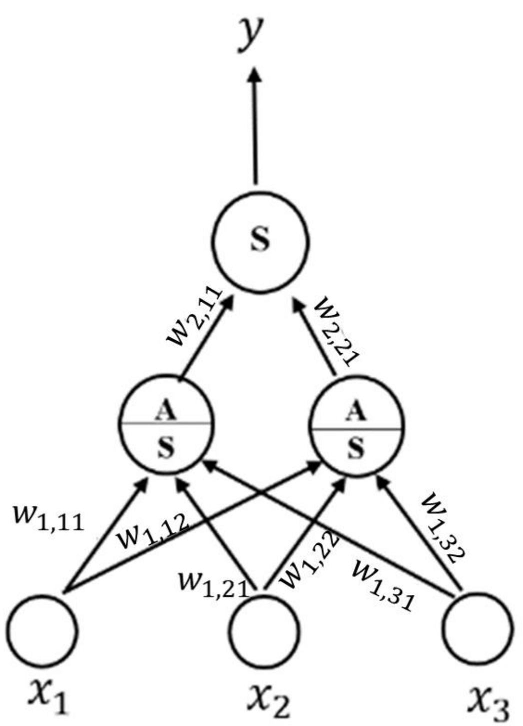
TRAINING OF NEURAL NETWORKS
STEEPEST DESCENT

- Example: Given sample: $x_1 = -3.2$, $x_2 = 4.1$, $x_3 = 0.163$ and $y = 2$
- Decision variables: 8 weights ($w_{k,lj}$)
- Objective function: $min_w \quad f = (\hat{y}\,(w) - \quad y)^2$
- Steepest descent update rule:

$$w_{k,lj}^{[q+1]} = w_{k,lj}^{[q]} - \alpha \left(\frac{\partial f}{\partial w_{k,lj}} \right)_{w^{[q]}}$$

- Evaluating gradient

$$\frac{\partial f}{\partial w_{k,lj}} = 2\,(\hat{y} - y)\, \frac{\partial \hat{y}}{\partial w_{k,lj}}$$



TRAINING OF NEURAL NETWORKS BACKPROPAGATION

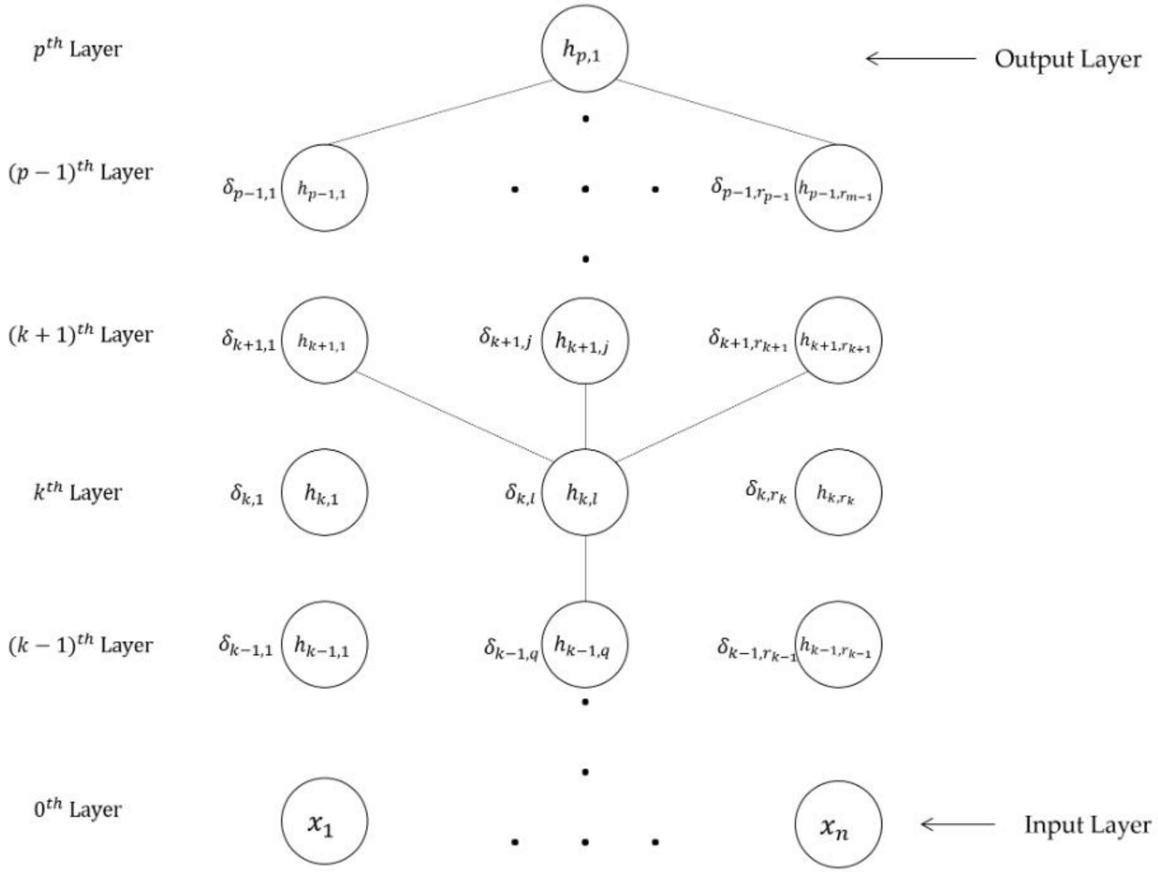
- Backpropagation – steepest descent with the gradient computed using chain rule of differentiation
- It efficiently computes one layer at a time, unlike a native direct computation
- Start with an initial guess of weights
- Each iteration has
  - Forward propagation (Given the weights, we calculate predicted output for the given input and error)
  - Backward propagation – Gradient calculation and updation of weights
- We will describe how the algorithm will work for one sample point; including all the points or a batch of points will only manifest as summation terms in the algorithm

TRAINING OF NEURAL NETWORKS BACKPROPAGATION

- Consider a p-layer neural network (0^{th} layer input, p^{th} layer output, p-1 hidden layers)
- Let the final output of the l^{th} node in k^{th} layer be $h_{k,l}$ and the intermediate output before the application of activation function be $o_{k,l}$
- r_k represents number of nodes in k^{rh} layer
- Predicted output $\hat{y} = h_{p,1}$
- Prediction error due to sample I

$$E^i = 0.5\left(y^i - \hat{y}^i\right)^2$$

- Total prediction error: $E = \sum E^i$



TRAINING OF NEURAL NETWORKS
BACKPROPAGATION

- For every node, we define a new variable, $\delta_{k,l}$
  - For the lth node in the kth layer

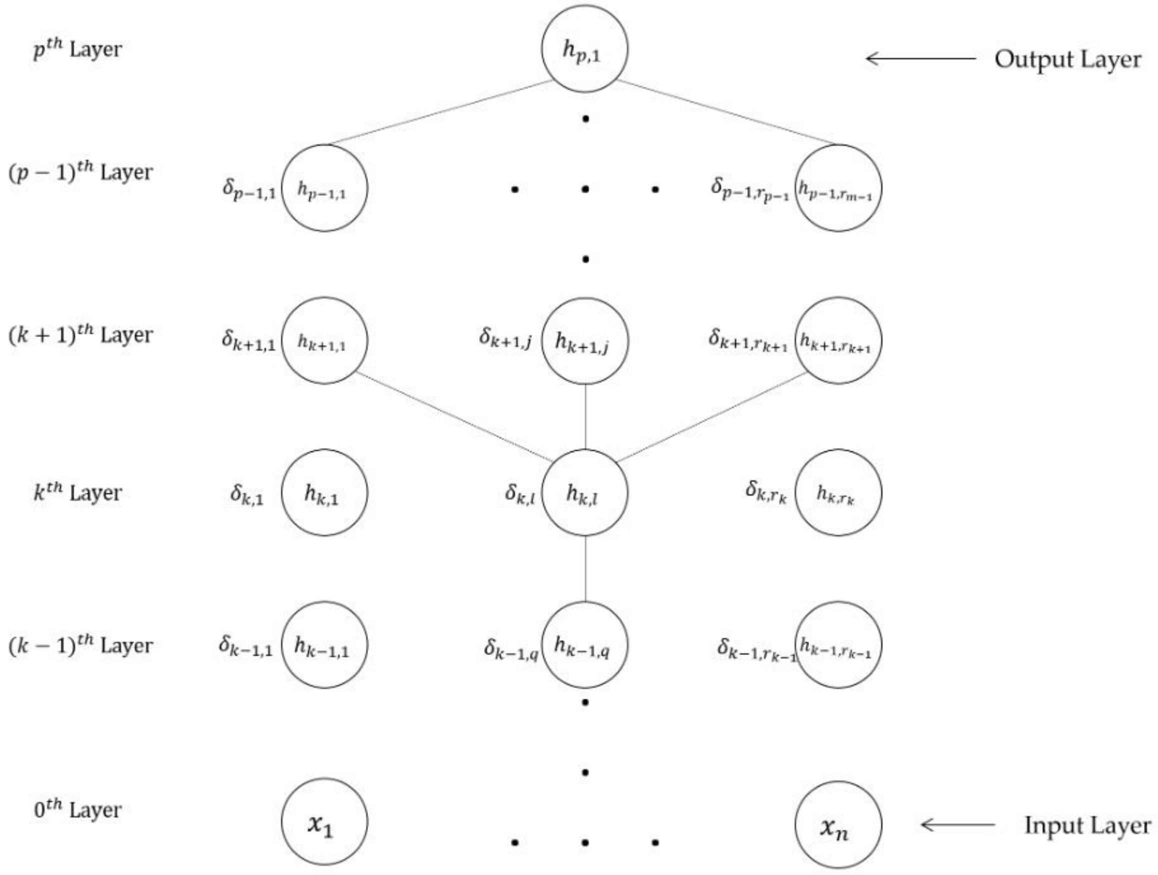
$$\delta_{k,l} = \frac{\partial E}{\partial o_{k,l}}$$

$$\delta_{k,l}^{i} = \frac{\partial E^{i}}{\partial o_{k,l}^{i}}$$

- Let the weight between lth node in the kth layer and jth node in the (k+1)th layer be $w_{k+1,lj}$
- Then,

$$h_{k,l}^{i} = A(o_{k,l}^{i}); \qquad h_{k+1,j}^{i} = A(o_{k+1,j}^{i})$$

$$o_{k+1,j}^{i} = \sum_{l=1}^{r_k} w_{k+1,lj} h_{k,l}^{i} = \sum_{l=1}^{r_k} w_{k+1,lj} A(o_{k,l}^{i})$$



TRAINING OF NEURAL NETWORKS BACKPROPAGATION

- To find $\delta_{k,l}^{i}$, we need to find $\frac{\partial E^i}{\partial o_{k,l}^i}$
- Changes in $o_{k,l}^i$ would affect $o_{k+1,j}^i$ which in turn affects E^i

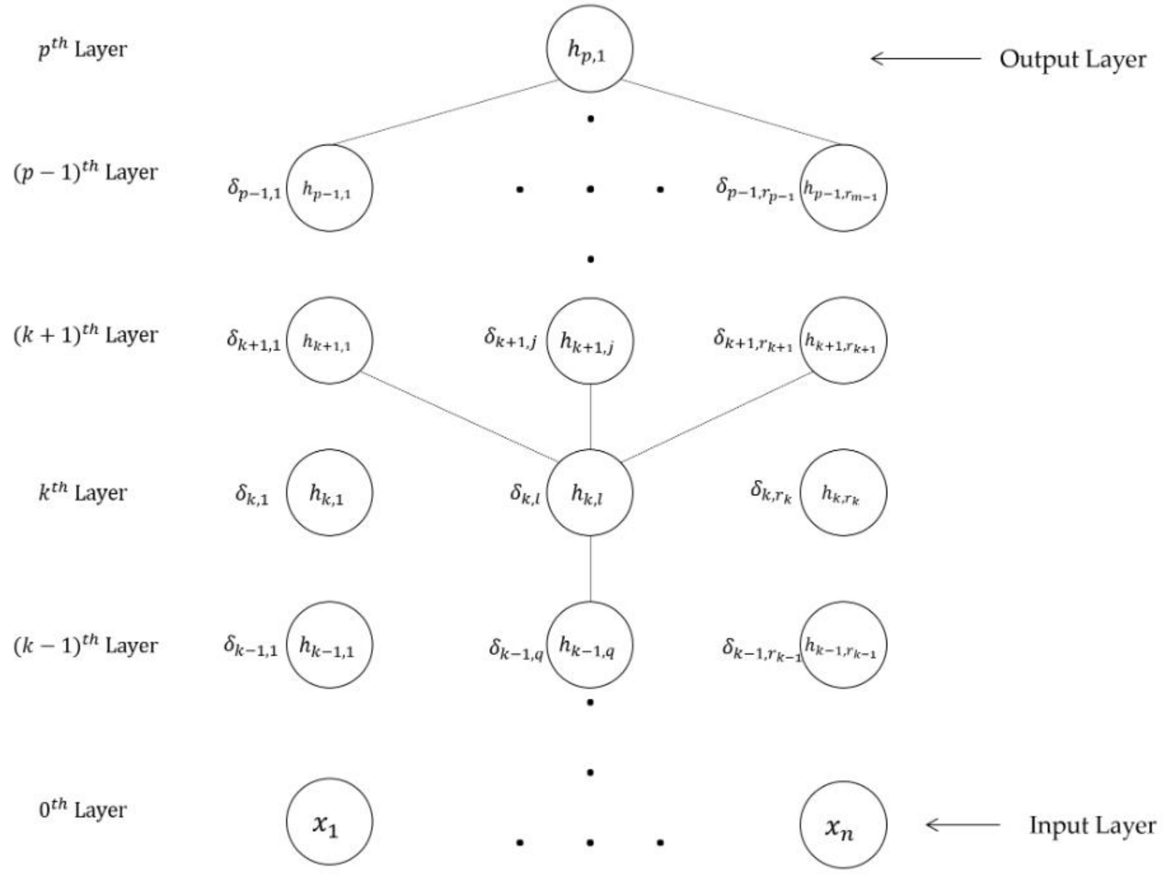
$$E^i = E^i\left(o_{k+1,j}^i\right)$$

- Using chain rule of differentiation,

$$\delta_{k,l}^{i} = \sum_{j=1}^{r_{k+1}} \frac{\partial E^i}{\partial o_{k+1,j}^i} \frac{\partial o_{k+1,j}^i}{\partial o_{k,l}^i} = \sum_{j=1}^{r_{k+1}} \delta_{k+1,j}^i \frac{\partial o_{k+1,j}^i}{\partial o_{k,l}^i}$$

- Since $\frac{\partial o_{k+1,j}^i}{\partial o_{k,l}^i} = w_{k+1,lj} A'(o_{k,l}^i)$

$$\delta_{k,l}^{i} = \sum_{j=1}^{r_{k+1}} \delta_{k+1,j}^i w_{k+1,lj} A'(o_{k,l}^i) = A'(o_{k,l}^i) \sum_{j=1}^{r_{k+1}} w_{k+1,lj} \delta_{k+1,j}^i$$



BACKPROPAGATION (m SAMPLES)

- For m samples (m could be entire samples or a batch),

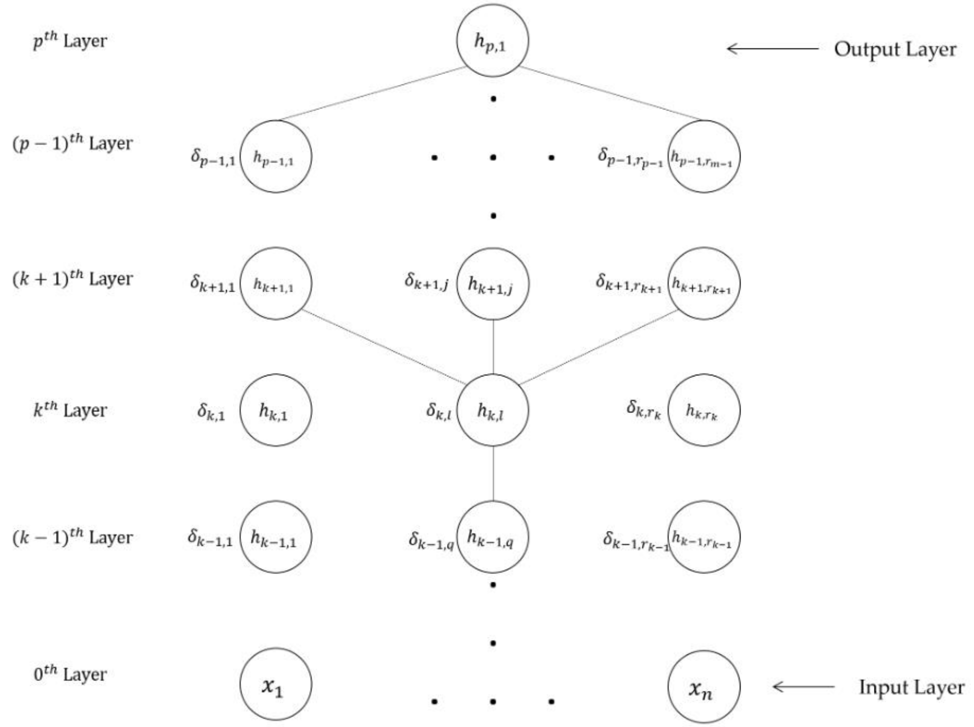
$$\delta_{k,l} = \frac{\partial E}{\partial o_{k,l}} = \sum_{i=1}^{m} \frac{\partial E^i}{\partial o_{k,l}^i} = \sum_{i=1}^{m} \delta_{k,l}^i = \sum_{i=1}^{m} \left(A'\left(o_{k,l}^i\right) \left(\sum_{j=1}^{r_{k+1}} w_{k+1,lj} \delta_{k+1,j}^i \right) \right)$$

$$\delta_{p,1} = \sum_{i=1}^{m} \delta_{p,1}^i = -\sum_{i=1}^{m} \left(y^i - h_{p,1}^i\right) A'\left(o_{p,1}^i\right)$$

- We first evaluate δ for the output node, then for all nodes in the (p-1)th layer, (p-2)th layer, and so on until the first layer
- Since we start at the output node and come all the way down to the input node, this scheme is referred to as back-propagation scheme

$$w_{k+1,lj}^n = w_{k+1,lj} + \eta \sum_{i=1}^{m} \left(\frac{-\partial E^i}{\partial w_{k+1,lj}} \right) = w_{k+1,lj} + \eta \sum_{i=1}^{m} \left(\frac{-\partial E^i}{\partial o_{k+1,j}} \right) \left(\frac{\partial o_{k+1,j}}{\partial w_{k+1,lj}} \right)$$

$$\implies w_{k+1,lj}^n = w_{k+1,lj} - \eta \sum_{i=1}^{m} \delta_{k+1,j}^i h_{k,l}^i \text{ (From 1.21)}$$

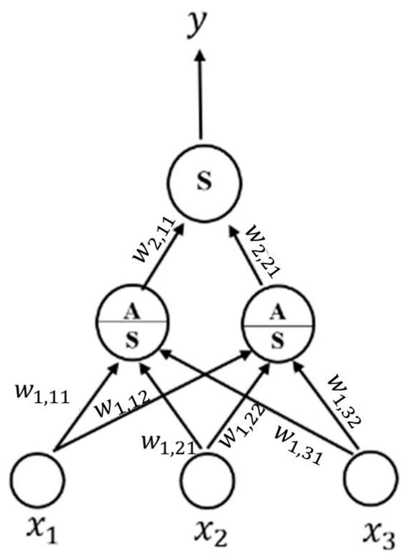


EXAMPLE

- Example: Given sample: $x_1 = -3.2$, $x_2 = 4.1$, $x_3 = 0.163$ and $y = 2$
- Decision variables: 8 weights ($w_{k,lj}$)
- Objective function: $min_w \quad f = (\hat{y}(w) - y)^2$
- Steepest descent update rule with back-propagation

$$w_{k+1,lj}^{q+1} = w_{k+1,lj}^{[q]} - \eta\, \delta_{k+1,j}^{[q]} h_{k,l}^{[q]}$$

Decision Variable	*Initial Guess*	*Iteration 1*
$w_{1,11}$	0.1	0.0633
$w_{1,21}$	0.3	0.347
$w_{1,31}$	0.5	0.5019
$w_{1,12}$	0.2	0.1602
$w_{1,22}$	0.4	0.4510
$w_{1,32}$	0.6	0.602
$w_{2,11}$	0.7	0.7605
$w_{2,21}$	0.8	0.8622
Error	0.8851	0.7059




operation == "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
_operation == "MIRROR_Y":
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
_operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
selection at the end -add
_ob.select= 1
ier_ob.select=1
ntext.scene.objects.active
("Selected" + str(modifier
irror_ob.select = 0
= bpy.context.selected_ob
ata.objects[one.name].se
int("please select exactl

THANKYOU