CS217: Artificial Intelligence and Machine Learning (associated lab: CS240)

Pushpak Bhattacharyya
CSE Dept.,
IIT Bombay

Week5 of 3feb25, sigmoid, softmax

Main points covered: week3 of 27jan25

Fundamental Observation

- The number of TFs computable by a perceptron is equal to the number of regions produced by 2^n hyper-planes,obtained by plugging in the values $<x_1,x_2,x_3,\ldots,x_n>$ in the equation

$$\sum_{i=1}^{n} w_i x_i = \theta$$

Number of regions founded by n hyperplanes in d-dim passing through origin is given by the following recurrence relation

$$R_{n,d} = R_{n-1,d} + R_{n-1,d-1}$$

we use generating function as an operating function

Boundary condition:

$R_{1,d} = 2$ 1 hyperplane in d-dim

$R_{n,1} = 2$ n hyperplanes in 1-dim,

Reduce to n points thru origin

The generating function is

$$f(x,y) = \sum_{n=1}^{\infty} \sum_{d=1}^{\infty} R_{n,d} \cdot x^n y^d$$

Comparing co-efficients we get

$$R_{n,d} = 2\sum_{i=0}^{d-1} C_i^{n-1}$$

Implication

- *R(n,d)* becomes for a perceptron with *m* weights and 1 threshold $R(2^m, m+1)$

$$= 2\sum_{i=0}^{m+1-1} C_i^{2^m-1}$$

$$= 2\sum_{i=0}^{m} C_i^{2^m-1}$$

$$= O(2^{m^2})$$

- Total no of Boolean Function is $2^{2^\wedge m}$. Shows why #TF << #BF

Gradient Descent Technique

- Let E be the error at the output layer

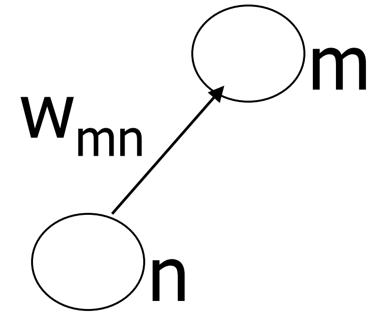
$$E = \frac{1}{2}\sum_{j=1}^{p}\sum_{i=1}^{n}(t_i - o_i)_j^2$$

- t_i = target output; o_i = observed output

- i is the index going over n neurons in the outermost layer
- j is the index going over the p patterns (1 to p)
- Ex: XOR:– p=4 and n=1

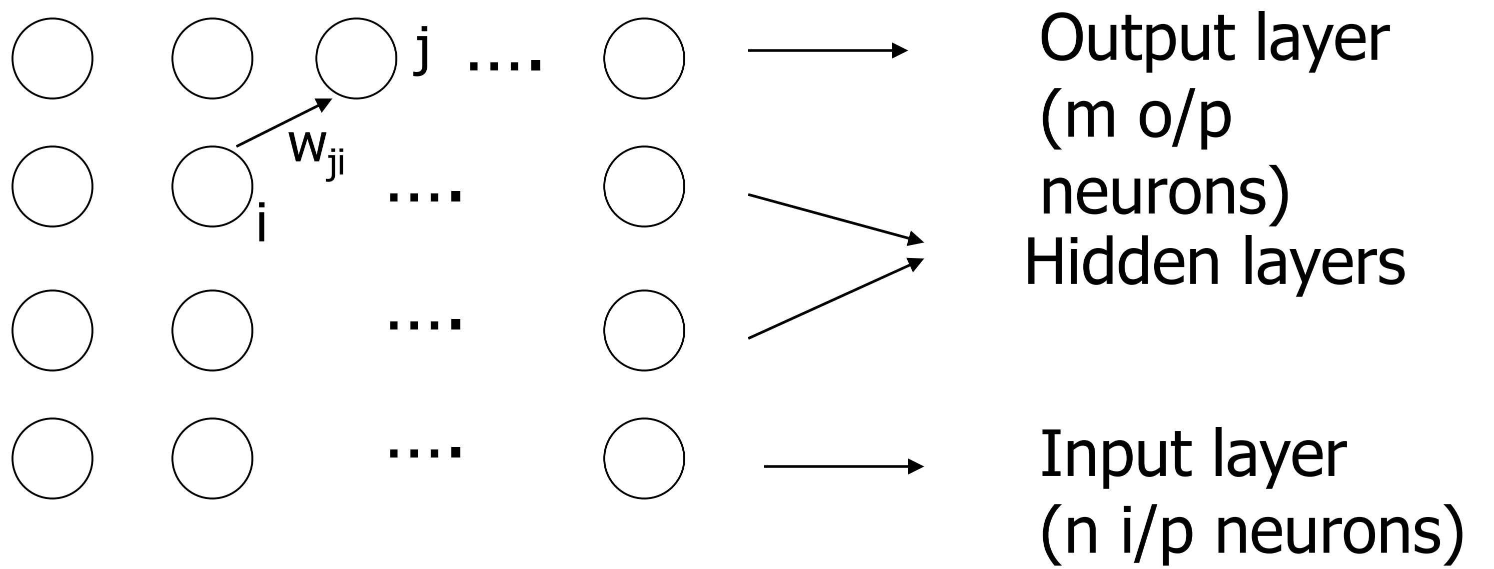
Weights in a FF NN

- w_{mn} is the weight of the connection from the n^{th} neuron to the m^{th} neuron
- E *vs* $\overline{W}$ surface is a complex surface in the space defined by the weights w_{ij}
- $-\frac{\delta E}{\delta w_{mn}}$ gives the direction in which a movement of the operating point in the w_{mn} co-ordinate space will result in maximum decrease in error



$$\Delta w_{mn} \propto -\frac{\delta E}{\delta w_{mn}}$$

Backpropagation algorithm



- Fully connected feed forward network
- Pure FF network (no jumping of connections over layers)

General Backpropagation Rule

- General weight updating rule:

$$\Delta w_{ji} = \eta \delta j o_i$$

- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) o_i \text{ for hidden layers}$$

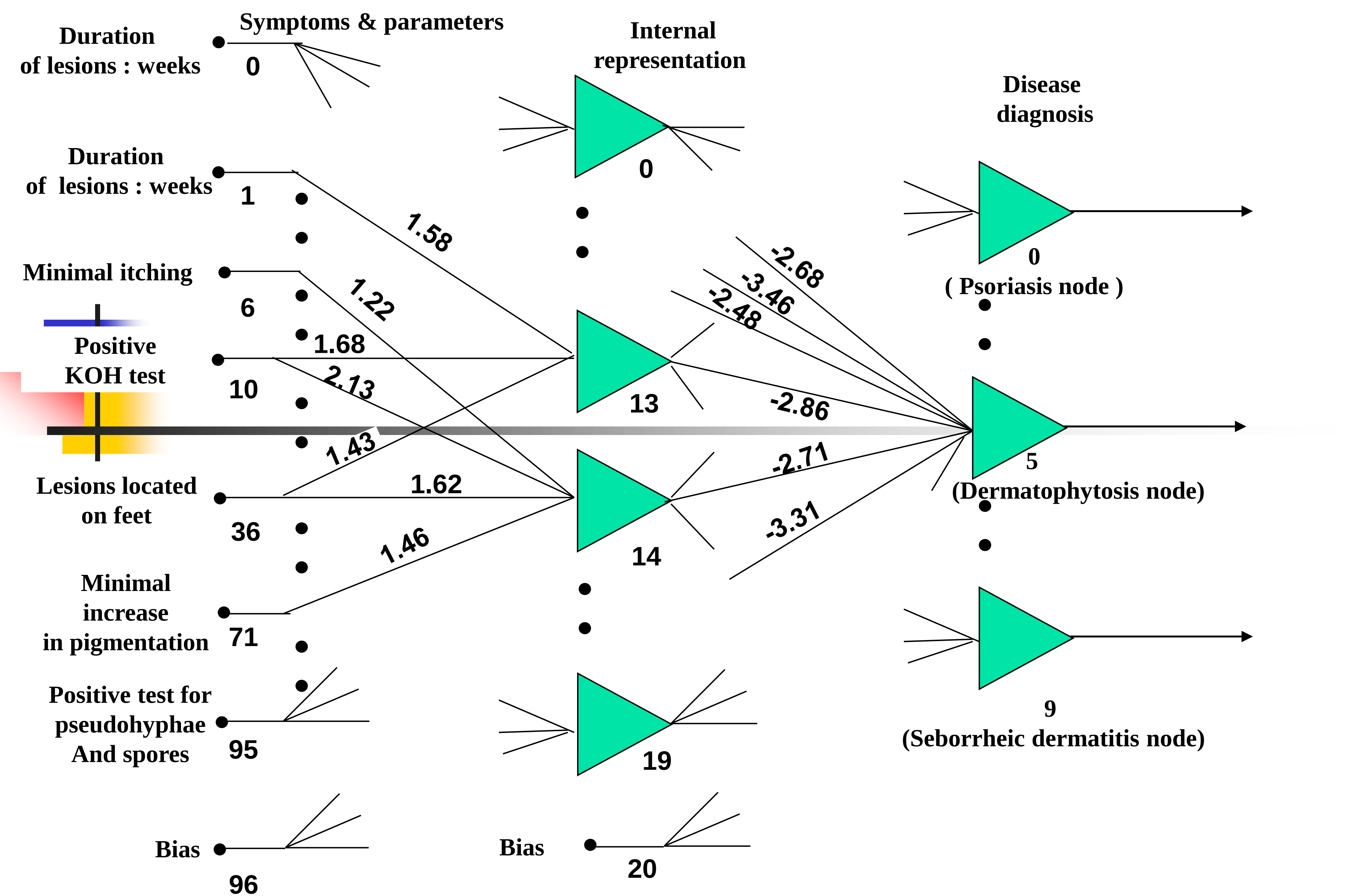
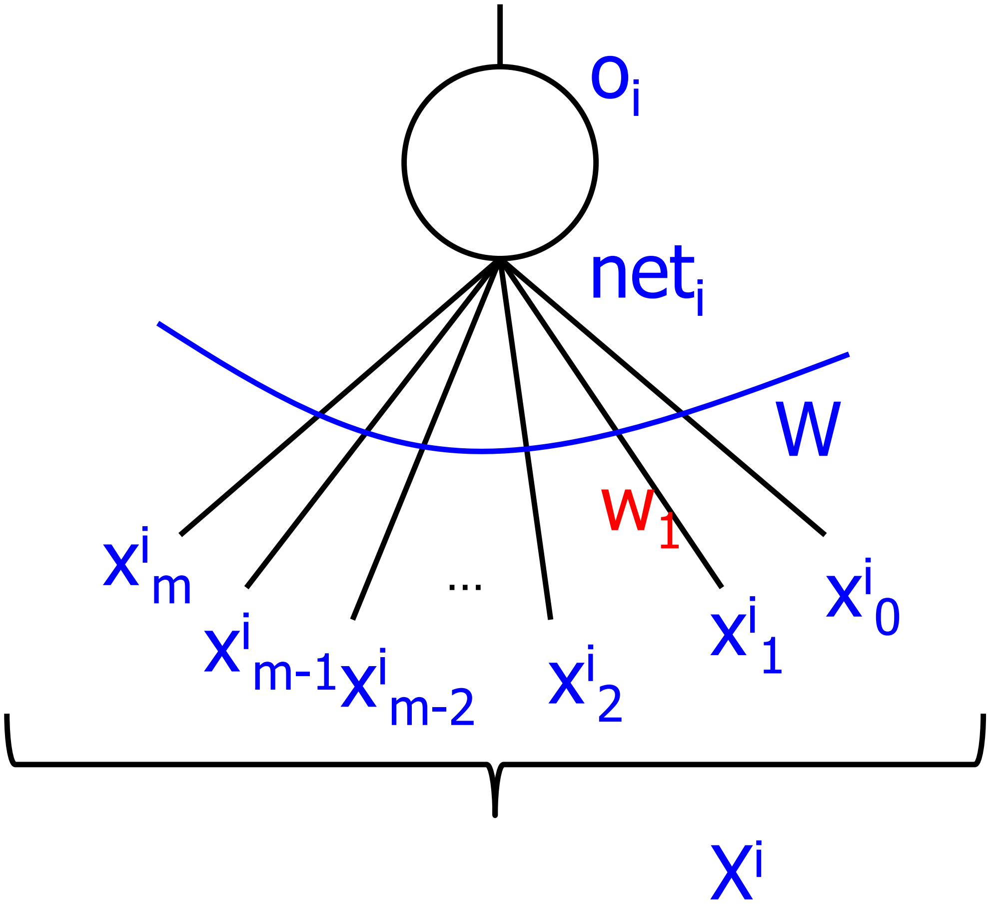


Figure : Explanation of dermatophytosis diagnosis using the DESKNET expert system.

End of main points

Sigmoid

Sigmoid neuron



$$o^i = \frac{1}{1 + e^{-net^i}}$$

$$net_i = W.X^i = \sum_{j=0}^{m} w_j x_j^i$$

Sigmoid function: can saturate

- Brain saving itself from itself, in case of extreme agitation, emotion etc.

Definition: Sigmoid or Logit function

$$y = \frac{1}{1+e^{-x}}$$

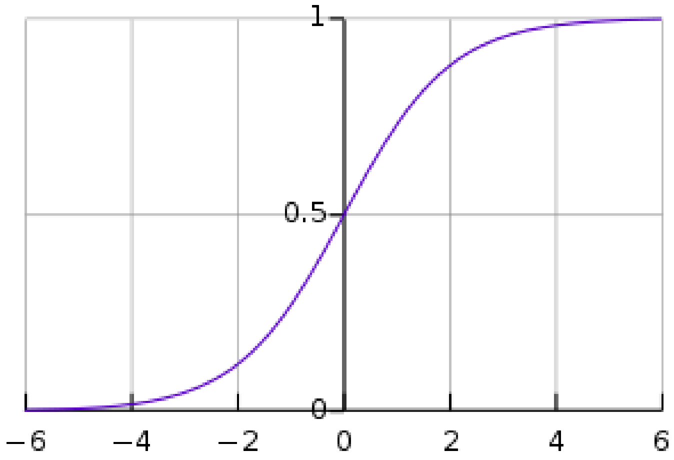
$$y = \frac{1}{1+e^{-kx}}$$

$$\frac{dy}{dx} = y(1-y)$$

$$\frac{dy}{dx} = ky(1-y)$$

If k tends to infinity, sigmoid tends to the step function

Sigmoid function



$$f(x) = \frac{1}{1+e^{-x}}$$

$$\begin{aligned}
f(x) &= \frac{1}{1+e^{-x}} \\
\frac{df(x)}{dx} &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\
&= \frac{e^{-x}}{(1+e^{-x})^{-2}} \\
&= \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) \\
&= f(x).(1 - f(x))
\end{aligned}$$

Decision making under sigmoid

- Output of sigmod is between 0-1
- Look upon this value as probability of Class-1 (C_1)
- *1-sigmoid(x)* is the probability of Class-2 (C_2)
- Decide C_1, if $P(C_1) > P(C_2)$, else C_2

Sigmoid function and multiclass classification

- Why can’t we use sigmoid for n-class classification? Have segments on the curve devoted to different classes, just like –infinity to 0.5 is for class 2 and 0.5 to plus infinity is class 2.

Output won't form mutually exclusive probability distribution if used for multiclass classification, as the output would be independent to each other no guarantee that their total sum equals one.

- Think about it!!

multiclass: SOFTMAX

- 2-class → multi-class (C classes)
- Sigmoid → softmax
- i^{th} input, c^{th} class (small c), c varies over classes
- In softmax, decide for that class which has the highest probability

  winner takes all.

What is softmax

- Turns a vector of *K* real values into a vector of *K* real values that sum to 1
- Input values can be positive, negative, zero, or greater than one
- But softmax transforms them into values between 0 and 1
- so that they can be interpreted as probabilities.

Mathematical form

$$\sigma(\bar{Z})_i = \frac{e^{Z_i}}{\sum_{j=1}^{K} e^{Z_j}}$$

- σ is the **softmax** function
- Z is the input vector of size K
- The RHS gives the i^{th} component of the output vector
- Input to softmax and output of softmax are of the same dimension

k-vector input --> k vector output.

Example

$$\bar{Z} = <1, 2, 3>$$

$$Z_1 = 1,\ Z_2 = 2,\ Z_3 = 3$$

$$e^1 = 2.72,\ e^2 = 7.39,\ e^3 = 20.09$$

$$\sigma(\bar{Z}) = < \frac{2.72}{2.72 + 7.39 + 20.09}, \frac{7.39}{2.72 + 7.39 + 20.09}, \frac{20.09}{2.72 + 7.39 + 20.09} >$$

$$= < .09, 0.24, 0.67 >$$

Softmax and Cross Entropy

- Intimate connection between softmax and cross entropy
- Softmax gives a vector of probabilities
- Winner-take-all strategy will give a classification decision

Winner-take-all with softmax

- Consider the softmax vector obtained from the example where the softmax vector is <0.09, 0.24, 0.65>
- These values correspond to 3 classes
  - For example, - *positive (+), negative (-)* and *neutral* (0) sentiments, given an input sentence like
  - *(a) I like the story line of the movie (+). (b) However the acting is weak (-). (c) The protagonist is a sports coach (0)*

Sentence vs. Sentiment

Sentence vs. Sentiment	**Positive**	**Negative**	**Neutral**
	(a) I like the story line of the movie (+). *(b) However the acting is weak (-).* *(c) The protagonist is a sports coach (0)*		
Sent (a)	1 *(P_{max} from softmax)*	0	0
Sentence (b)	0	1 *(P_{max} from softmax)*	0
Sentence (C)	0	0`	1 *(Pmax from softmax)*

Training data

- *(a) I like the story line of the movie (+).*
- *(b) However the acting is weak (-).*
- *(c) The protagonist is a sports coach (0)*

Input	Output
(a)	<1,0,0>
(b)	<0,1,0>
(c)	<0,0,1>

Finding the error

- Difference between target (T) and obtained (Y)
- Difference is called **LOSS**
- Options:
  - Total Sum Square Loss (TSS)
  - Cross Entropy *(measures difference between two probability distributions)*
- Softmax goes with cross entropy

Cross Entropy Function

observed output (mostly after passing through SoftMax)

labeled output

$$H(P,Q) = -\sum_{x=1,N}\sum_{k=1,C} P(x,k)\log_2 Q(x,k)$$

x varies over *N* data instances, *c* varies over *C* classes
P is target distribution; *Q* is observed distribution

Cross Entropy Loss

- Can we sum up cross entropies over the instances? Is it allowed?

- Yes, summing up cross entropies (i.e. the total cross entropy loss) is equivalent to multiplying probabilities.

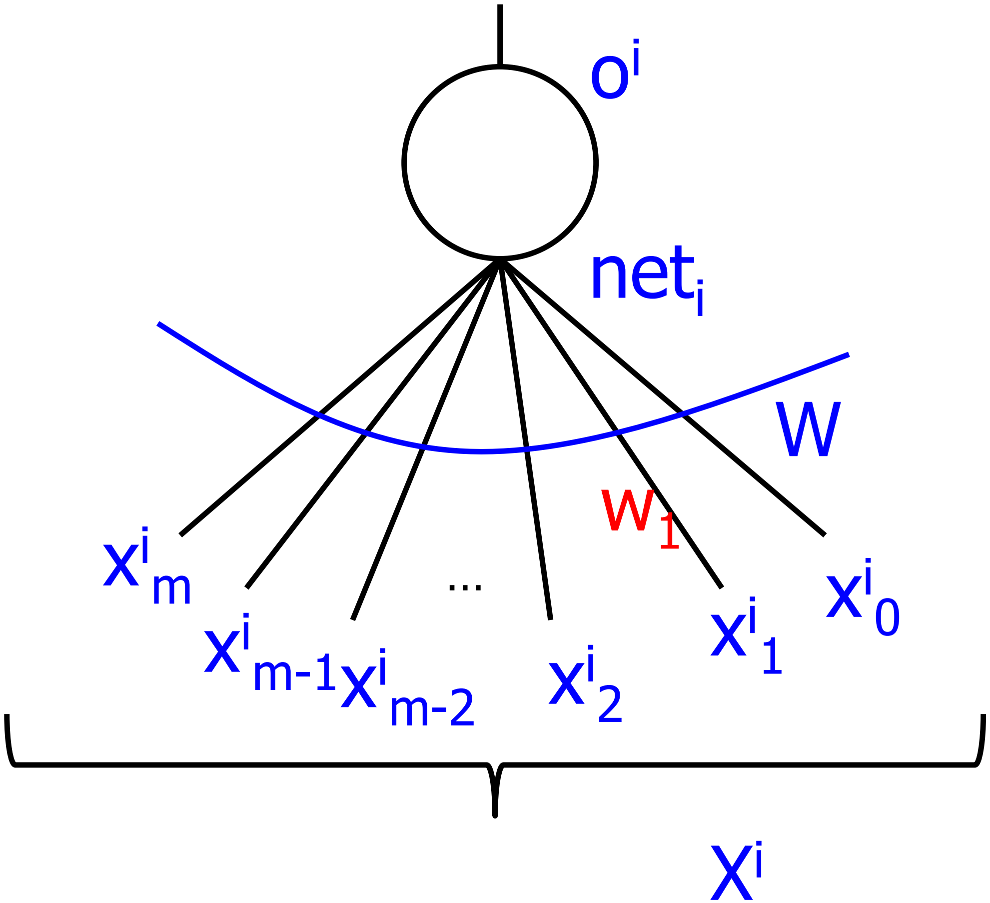
- Minimizing the total cross entropy loss is equivalent to maximizing the likelihood of observed data.

How to minimize loss

- Gradient descent approach
- Backpropagation Algorithm
- Involves derivative of the input-output function for each neuron
- FFNN with BP is the most important TECHNIQUE for us in the course

Sigmoid and Softmax neurons

Sigmoid neuron

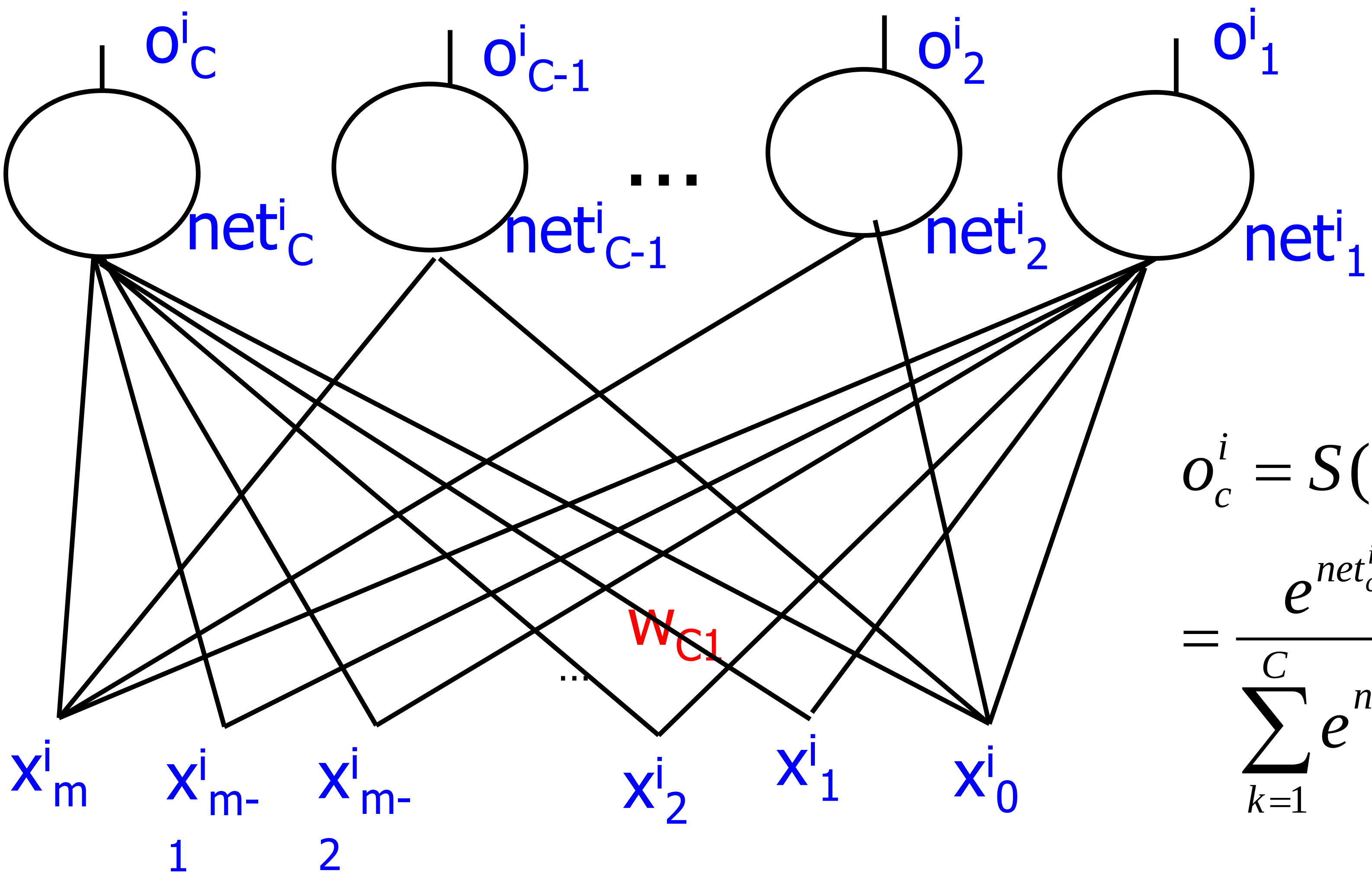


$$o^i = \frac{1}{1+e^{-net^i}}$$

$$net_i = W.X^i = \sum_{j=0}^{m} w_j x_j^i$$

Softmax Neuron

actual output is produced after processing through softmax.



$$o_c^i = S(NET^i)_c$$

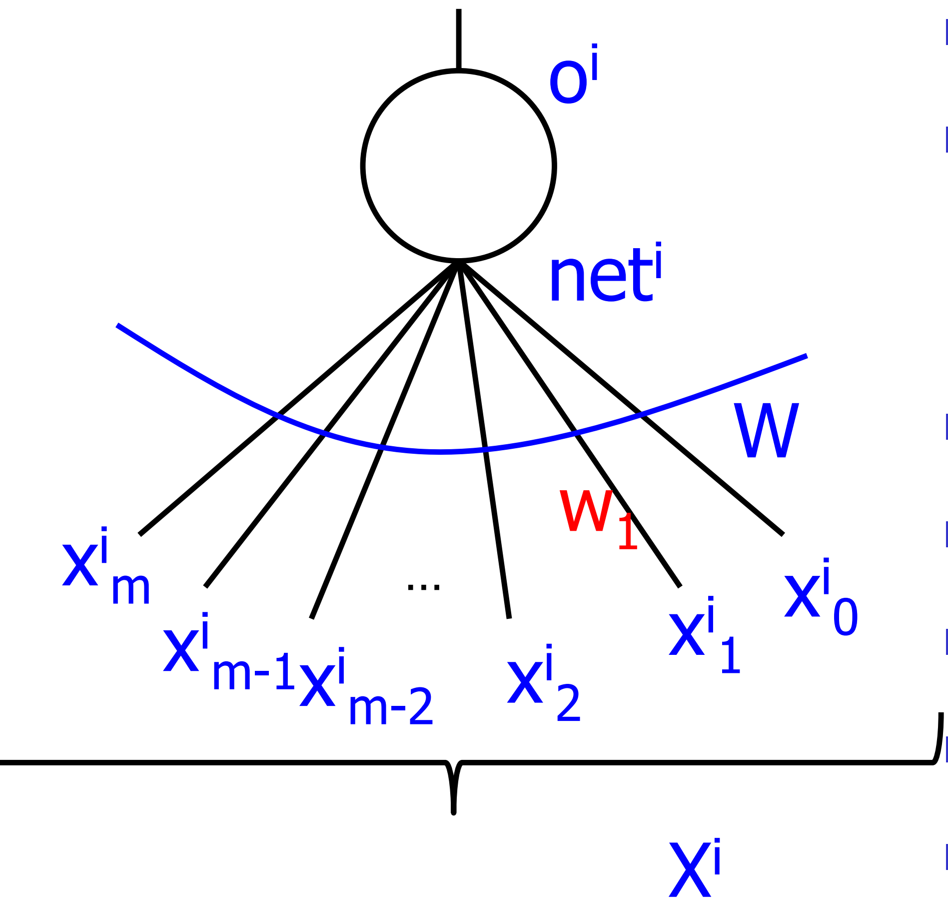
$$= \frac{e^{net_c^i}}{\sum_{k=1}^{C} e^{net_k^i}}$$

Output for class c (small c), c:1 to C

Notation

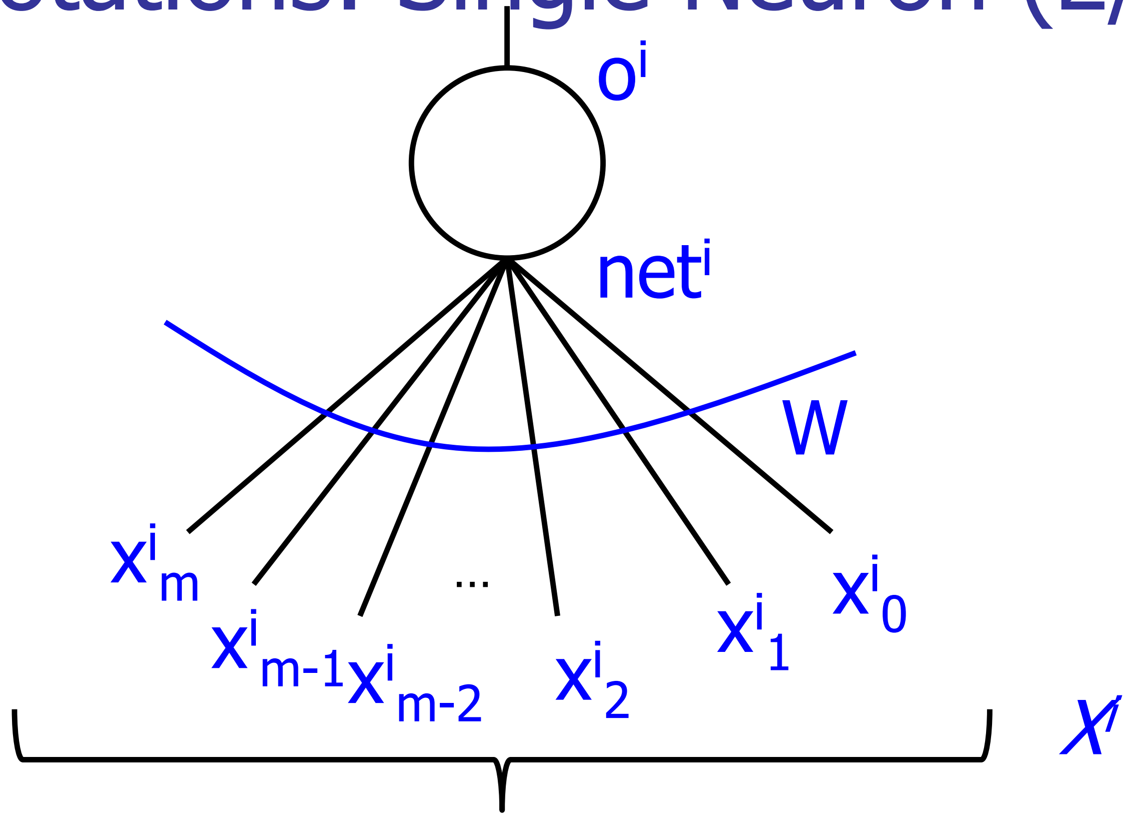
- $i=1..N$
- N i-o pairs, i runs over the training data
- $j=0...m$, m components in the input vector, j runs over the input dimension (also weight vector dimension)
- $k=1...C$, C classes (C components in the output vector)

Fix Notations: Single Neuron (1/2)



- Capital letter for vectors
- Small letter for scalars (therefore for vector components)
- X^i: i^{th} input vector
- o_i: output (scalar)
- W: weight vector
- net_i: $W.X^i$
- There are n input-output observations

Fix Notations: Single Neuron (2/2)



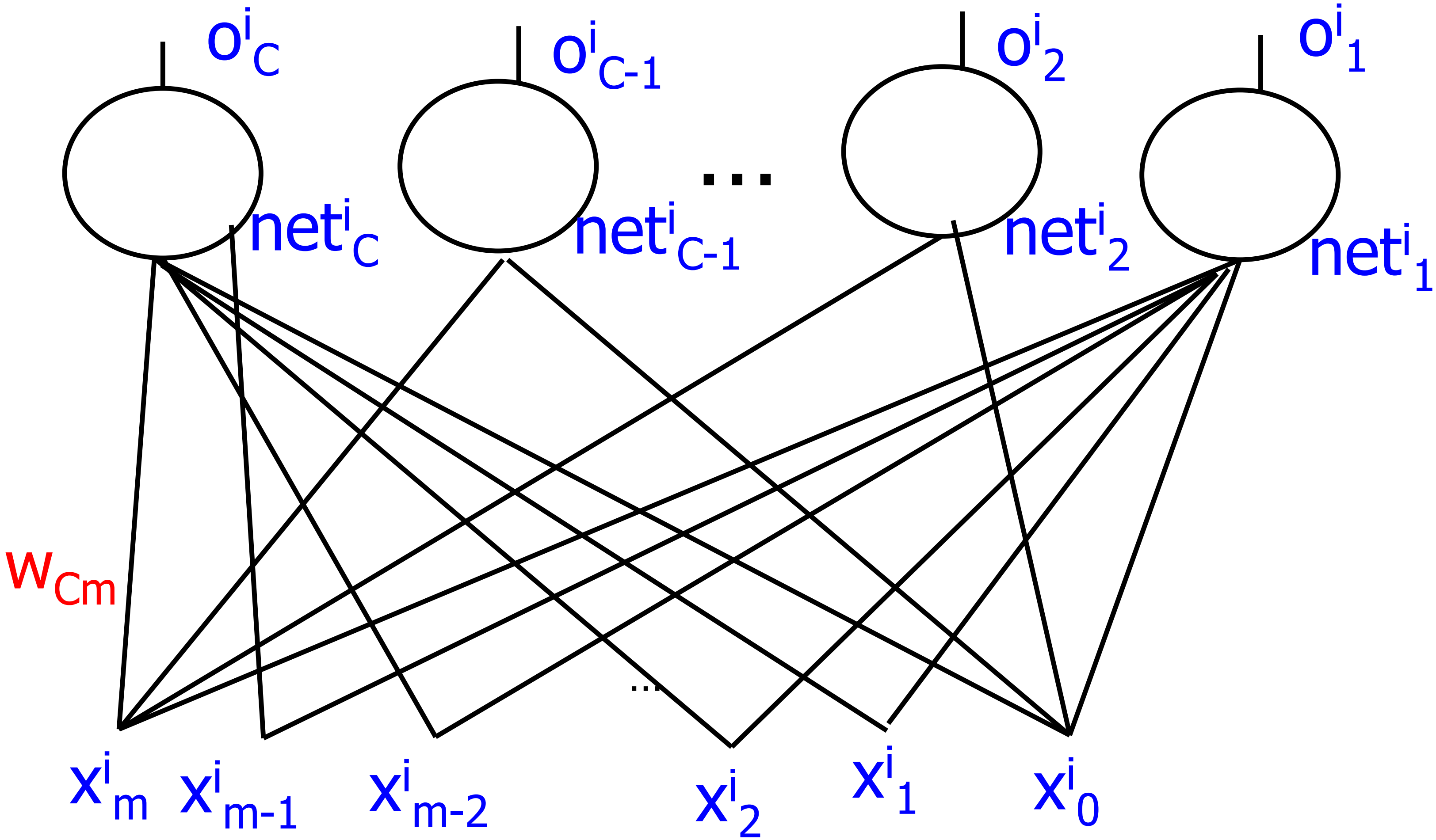
W and each X^i has *m* components

W: $<w_m, w_{m-1}, \ldots, w_2, w_0>$

X^i: $<x^i_m, x^i_{m-1}, \ldots, x^i_2, x^i_0>$

Upper suffix *i* indicates i^{th} input

Fixing Notations: Multiple neurons in o/p layer

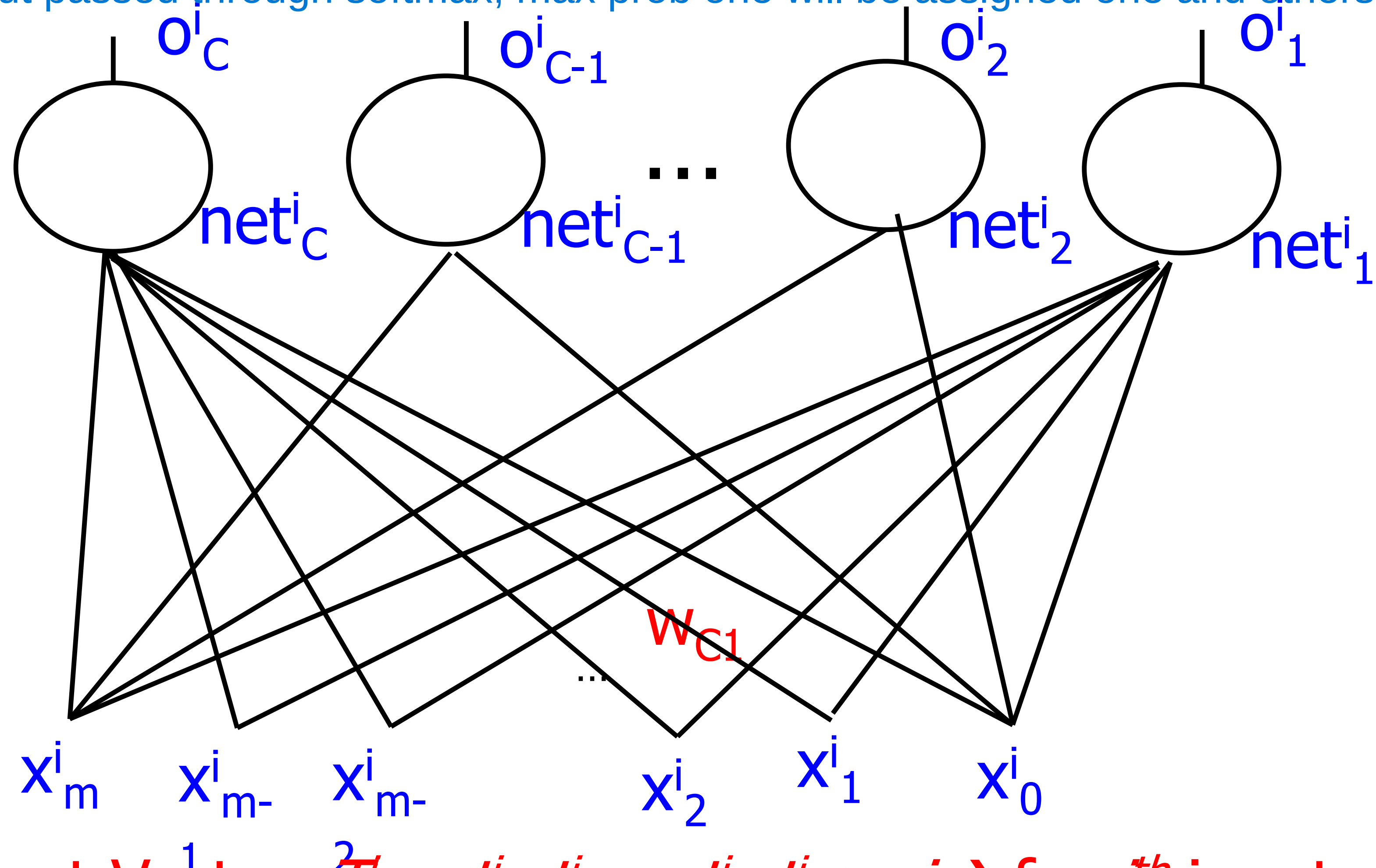


Now, O^i and NET^i are vectors for i^{th} input

W_c is the weight vector for c^{th} output neuron, $c=1..C$

Fixing Notations

net input passed through softmax, max prob one will be assigned one and others zero.



Target Vector, T^i: $<t^i_C\ t^i_{C-1} \ldots t^i_2\ t^i_1>$, $i \rightarrow$ for i^{th} input.

Only one of these C componets is 1, rest are 0

Derivatives

Derivative of sigmoid

$$o^i = \frac{1}{1+e^{-net^i}}, \; for \; i^{th} \; input$$

$$\ln o^i = -\ln(1+e^{-net^i})$$

$$\frac{1}{o^i}\frac{\partial o^i}{\partial net^i} = -\frac{1}{1+e^{-net^i}} . -e^{-net^i} = \frac{e^{-net^i}}{1+e^{-net^i}} = (1-o^i)$$

$$\Rightarrow \frac{\partial o^i}{\partial net^i} = o^i(1-o^i)$$

Derivative of Softmax

$$o_c^i = \frac{e^{net_c^i}}{\sum_{k=1}^{C} e^{net_k^i}}, \; i^{th} \; input \; pattern$$

Derivative of Softmax: Case-1, class *c* for O and NET same

$$\ln o_c^i = net_c^i - \ln\left(\sum_{k=1}^{C} e^{net_k^i}\right)$$

$$\frac{1}{o_c^i}\frac{\partial o_c^i}{\partial net_c^i} = 1 - \frac{1}{\sum_{k=1}^{C} e^{net_k^i}}.e^{net_c^i} = 1 - o_c^i$$

$$\Rightarrow \frac{\partial o_c^i}{\partial net_c^i} = o_c^i(1 - o_c^i)$$

Derivative of Softmax: Case-2, class *c′* in *net*$^i_{c'}$ different from class c of O

$$\ln o_c^i = net_c^i - \ln\left(\sum_{k=1}^{C} e^{net_k^i}\right)$$

$$\frac{1}{o_c^i}\frac{\partial o_c^i}{\partial net_{c'}^i} = 0 - \frac{1}{\sum_{k=1}^{C} e^{net_k^i}}.e^{net_{c'}^i} = -o_{c'}^i$$

$$\Rightarrow \frac{\partial O_c^i}{\partial net_{c'}^i} = -o_c^i o_{c'}^i$$

Finding weight change rule

Foundation: Gradient descent

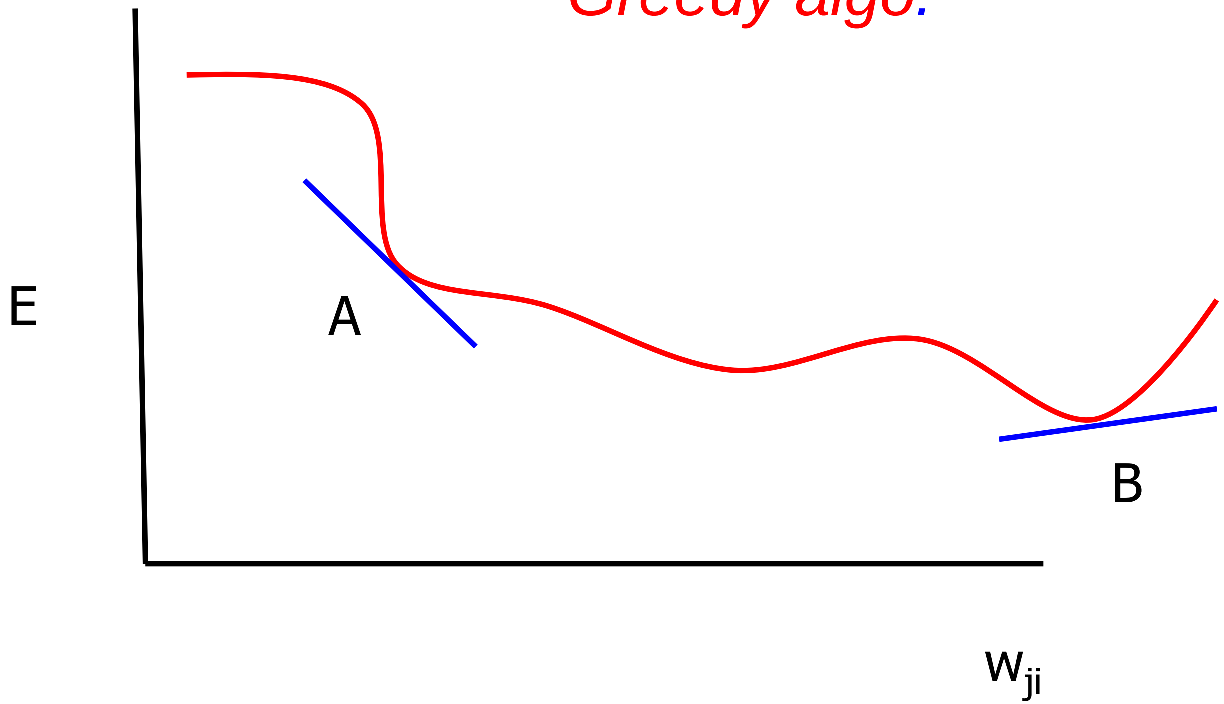
Change is weight $\Delta w_{ji} = -\eta \delta L / \delta w_{ji}$

η= *learning rate,*
L=loss, w_{ji}= *weight of connection from the* i^{th} *neuron to* j^{th}

At A, $\delta L / \delta w_{ji}$ *is negative, so* Δw_{ji} *is positive.*

At B, $\delta L / \delta w_{ji}$ *is positive, so* Δw_{ji} *is negative.*

E always decreases. Greedy algo.



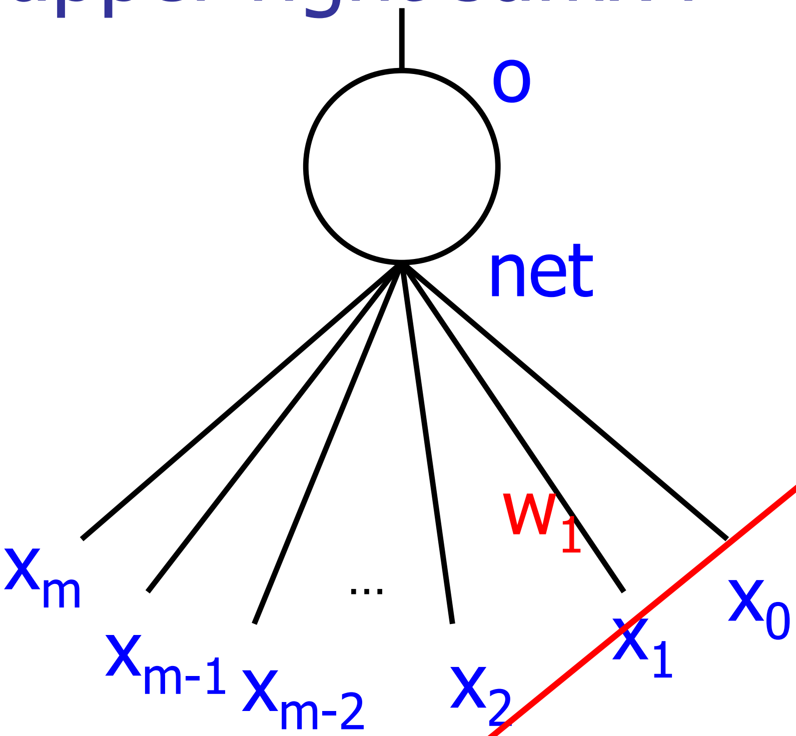
Gradient Descent is Greedy!

- Gradient Descent is greedy- always moves in the direction of reducing error
- Probabilistically also move in the direction of increasing error, to be able to come out of local minimum
- Nature randomly introduces some variation, and a totally new species emerges
- Darwin’s theory of evolution

Genetic Algorithm

- Genetic Algorithms: adaptive heuristic search algorithms
- used to generate high-quality solutions for optimization problems and search problems
- To evolve the generation, genetic algorithms use the following operators, all PROBABILSTICALLY
  - Selection, Cross over, Mutation

Single sigmoid neuron and *cross entropy* loss, derived for single data point, hence dropping upper right suffix *i*



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial net} \cdot \frac{\partial net}{\partial w_1}$$

$$L = -t \log o - (1-t)\log(1-o)$$

$$\Rightarrow \frac{\partial L}{\partial o} = -\frac{t}{o} + \frac{1-t}{1-o} = -\frac{t-o}{o(1-o)}$$

$$o = \frac{1}{1+e^{-net}} \Rightarrow \frac{\partial o}{\partial net} = o(1-o)$$

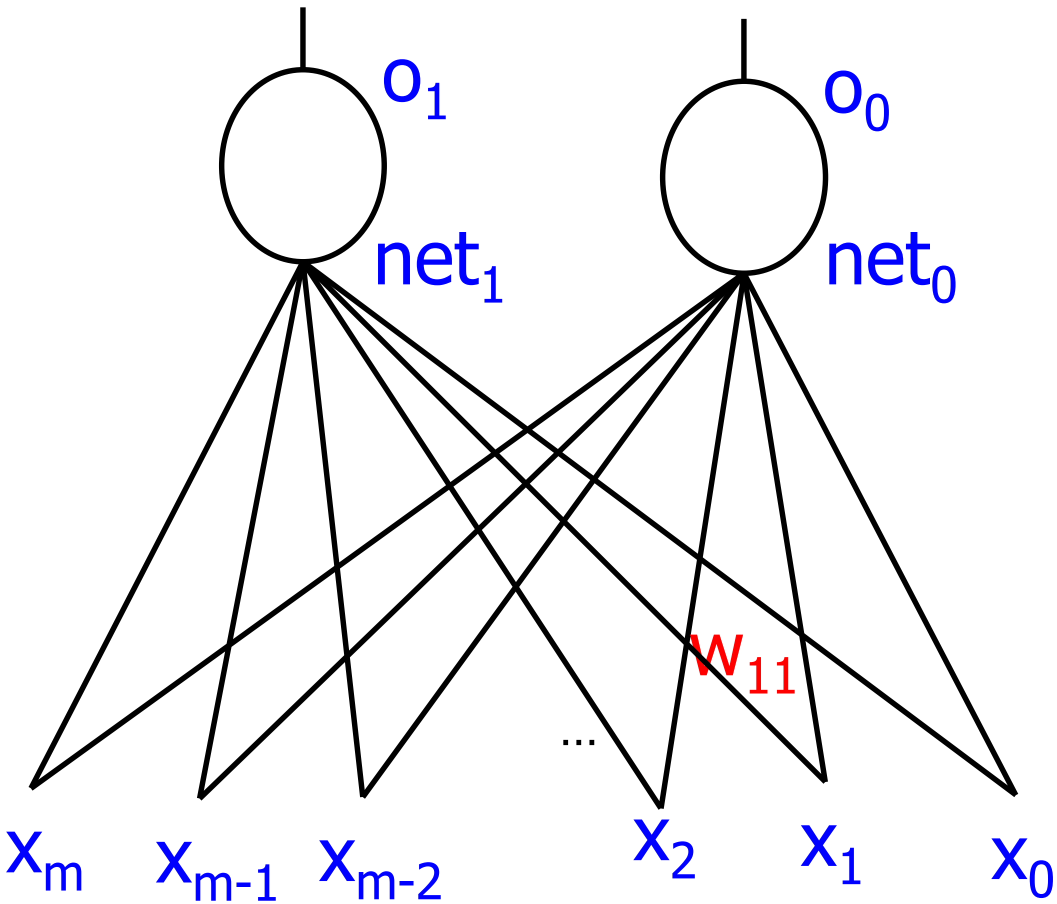
$$net = \sum_{j=0}^{m} w_j x_j \Rightarrow \frac{\partial net}{\partial w_1} = x_1$$

$$\Rightarrow \Delta w_1 = \eta \frac{\partial L}{\partial w_1} = \eta(t-o)x_1$$

SoftMax for single neuron.

$$\Delta w_1 = \eta(t-o)x_1$$

Multiple neurons in the output layer: softmax+*cross entropy* loss (1/2): illustrated with 2 neurons and single training data point



$$O = < o_1, o_0 >$$

$$NET = < net_1, net_0 >$$

$$o_1 = \frac{e^{net_1}}{e^{net_1} + e^{net_0}},\ o_0 = \frac{e^{net_0}}{e^{net_1} + e^{net_0}}$$

$$\frac{\partial O}{\partial NET} = \begin{bmatrix} \frac{\partial o_0}{\partial net_0} & \frac{\partial o_1}{\partial net_0} \\ \frac{\partial o_0}{\partial net_1} & \frac{\partial o_1}{\partial net_1} \end{bmatrix}$$

$$= \begin{bmatrix} o_0(1 - o_0) & -o_0 o_1 \\ -o_1 o_0 & o_1(1 - o_1) \end{bmatrix}$$

Softmax and Cross Entropy (2/2)

$$L = -t_1 \log o_1 - t_0 \log o_0$$

$$o_1 = \frac{e^{net_1}}{e^{net_1} + e^{net_0}},\ o_0 = \frac{e^{net_0}}{e^{net_1} + e^{net_0}}$$

$$\frac{\partial L}{\partial w_{11}} = -\frac{t_1}{o_1}\frac{\partial o_1}{\partial w_{11}} - -\frac{t_0}{o_0}\frac{\partial o_0}{\partial w_{11}}$$

$$\frac{\partial o_1}{\partial w_{11}} = \frac{\partial o_1}{\partial net_1}.\frac{\partial net_1}{\partial w_{11}} + \frac{\partial o_1}{\partial net_0}.\frac{\partial net_0}{\partial w_{11}} = o_1(1-o_1)x_1 + 0$$

$$\frac{\partial o_0}{\partial w_{11}} = \frac{\partial o_0}{\partial net_1}.\frac{\partial net_1}{\partial w_{11}} + \frac{\partial o_0}{\partial net_0}.\frac{\partial net_0}{\partial w_{11}} = -o_1 o_0 x_1 + 0$$

$$\Rightarrow \frac{\partial L}{\partial w_{11}} = -t_1(1-o_1)x_1 + t_0 o_1 x_1 = -t_1(1-o_1)x_1 + (1-t_1)o_1 x_1$$

$$= [-t_1 + t_1 o_1 + o_1 - t_1 o_1]x_1 = -(t_1 - o_1)x_1$$

$$\Delta w_{11} = -\eta \frac{\partial E}{\partial w_{11}} = \eta(t_1 - o_1)x_1$$

no contribution of other neurons input output needed.

Can be generalized

- When *L* is Cross Entropy Loss, the change in any weight is

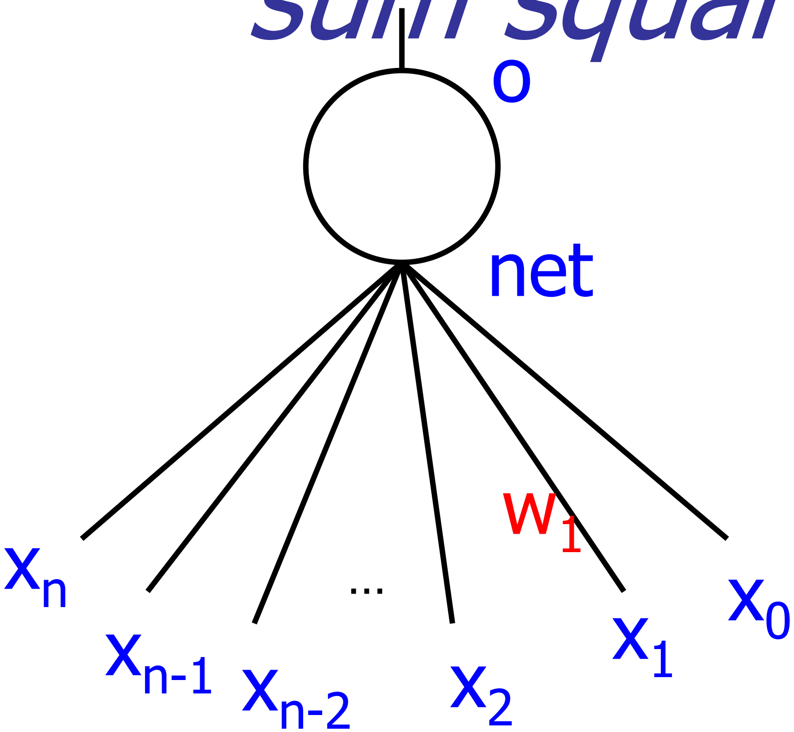
  learning rate *

  diff between target and observed outputs *

  input at the connection

Weight change rule with TSS

Single neuron: *sigmoid+total sum square* (tss) loss



Lets consider wlg w_1. Change is weight $\Delta w_1 = -\eta \delta L / \delta w_1$

η= learning rate,

L=loss= ½(t-o)²,

t=target, o=observed output

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial net} \cdot \frac{\partial net}{\partial w_1}$$

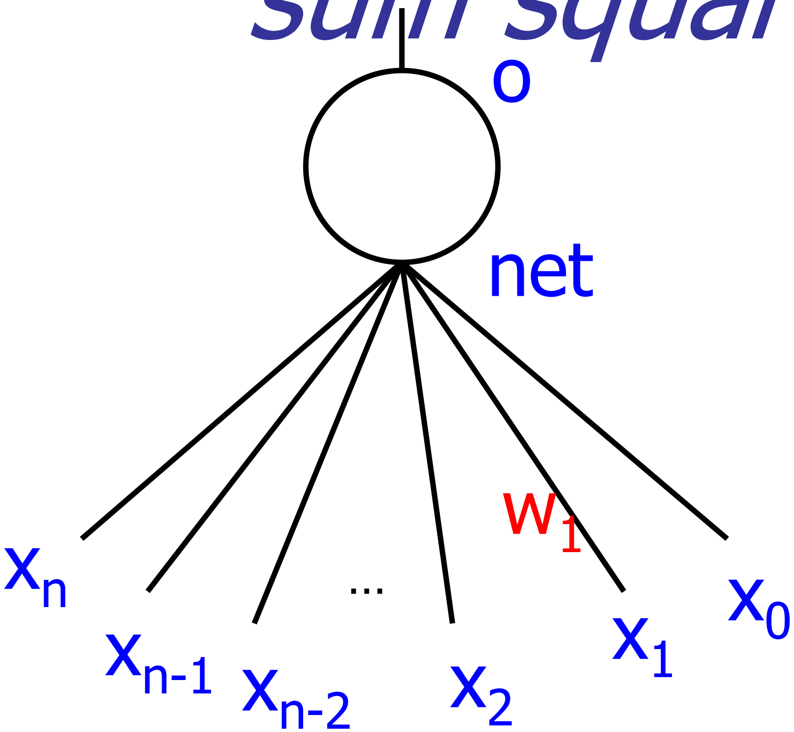
$$L = \frac{1}{2}(t-o)^2 \;\Rightarrow \frac{\partial L}{\partial o} = -(t-o)$$

$$o = \frac{1}{1+e^{-net}}\,(sigmoid) \Rightarrow \frac{\partial o}{\partial net} = o(1-o)$$

$$net = \sum_{i=0}^{n} w_i x_i \;\Rightarrow \frac{\partial net}{\partial w_1} = x_1$$

$$\Rightarrow \Delta w_1 = \eta(t-o)o(1-o)x_1$$

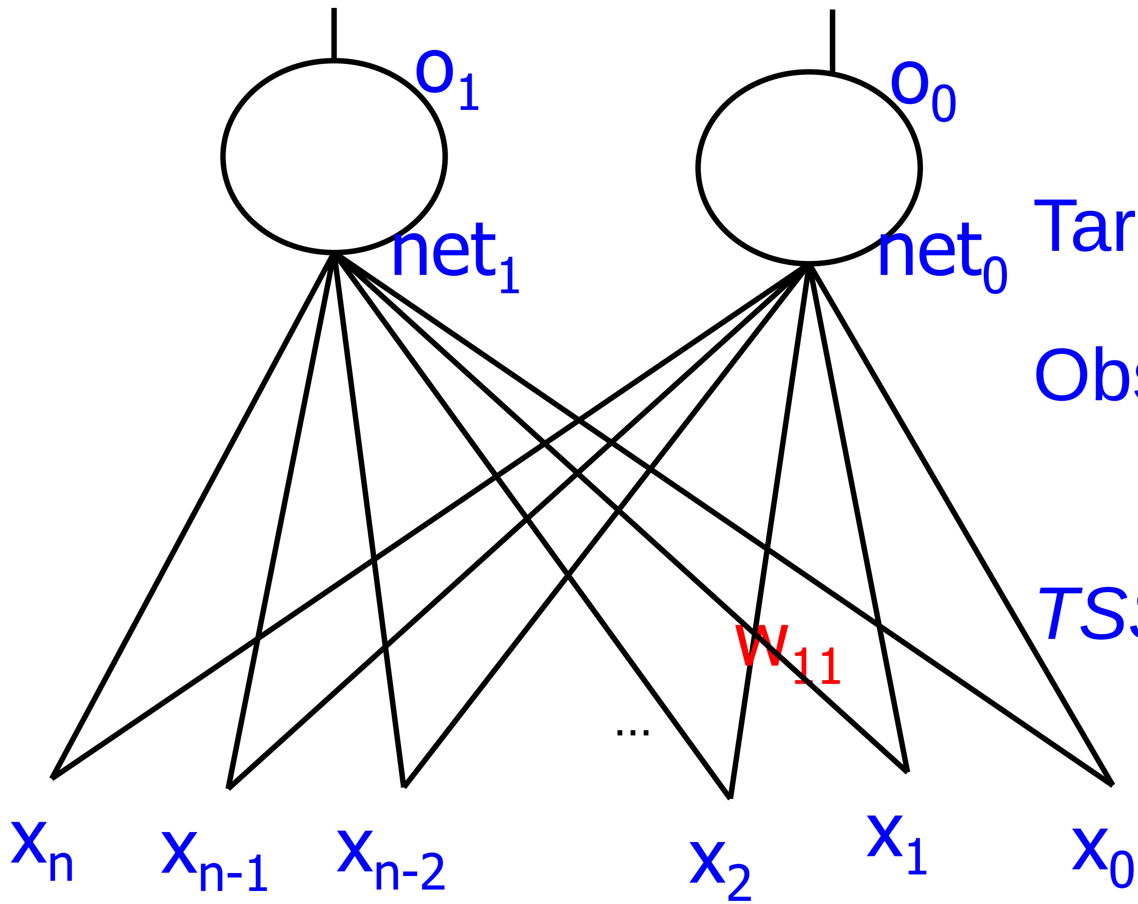
Single neuron: *sigmoid+total sum square* (tss) loss (cntd)



$$\Delta w_1 = \eta(t-o)o(1-o)x_1$$

Multiple neurons in the output layer: *sigmoid+total sum square* (tss) loss

Total sum square we don't use it with softmax.



Target vector: $\langle t_1, t_0 \rangle$

Observed vector: $\langle o_1, o_0 \rangle$

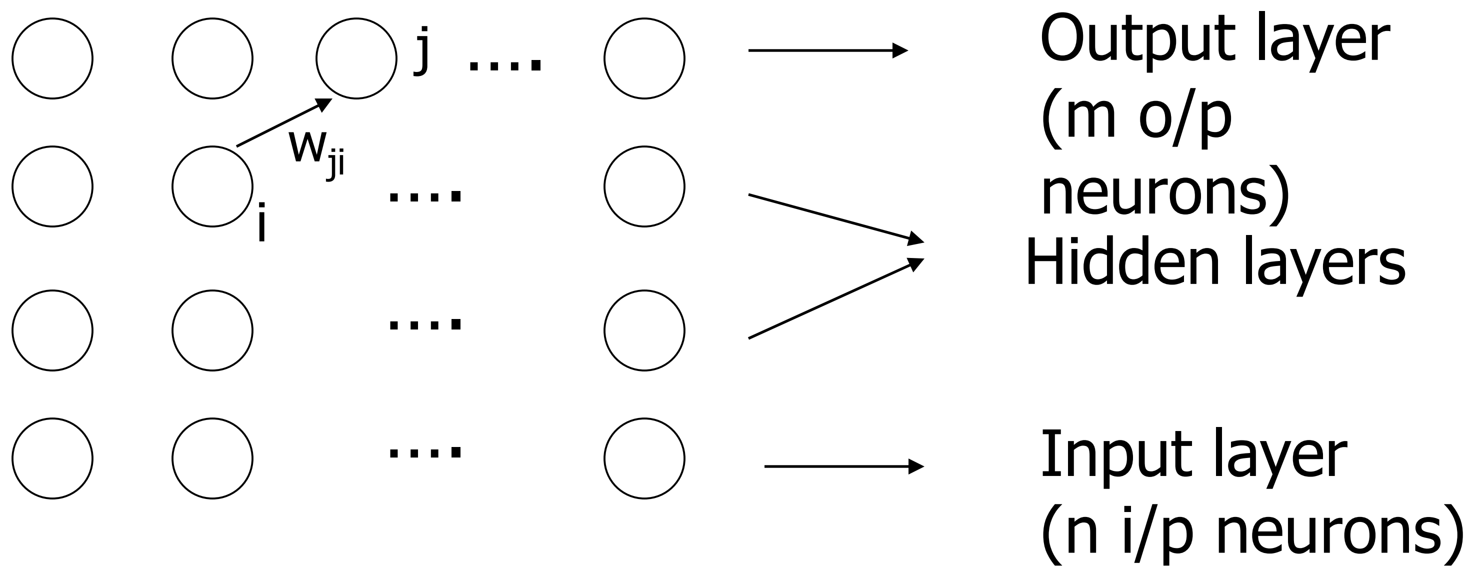
TSS Loss, $L = \frac{1}{2}[(t_1-o_1)^2 + (t_0-o_0)^2]$

$$\Delta w_{11} = \eta(t_1-o_1)o_1(1-o_1)x_1$$

Backpropagation

With total sum square loss (TSS)

Backpropagation algorithm



- Fully connected feed forward network
- Pure FF network (no jumping of connections over layers)

Gradient Descent Equations

$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}} (\eta = \text{learning rate}, 0 \le \eta \le 1)$$

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}} (net_j = \text{input at the j}^{th} \text{ neuron})$$

$$\frac{\delta E}{\delta net_j} = -\delta j$$

$$\Delta w_{ji} = \eta \delta j \frac{\delta net_j}{\delta w_{ji}} = \eta \delta j o_i$$

A quantity of great importance

Backpropagation – for outermost layer

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j} (net_j = \text{input at the j}^{th} \text{ layer})$$

$$E = \frac{1}{2}\sum_{j=1}^{N}(t_j - o_j)^2$$

$$\text{Hence, } \delta j = -(-(t_j - o_j)o_j(1-o_j))$$

$$\Delta w_{ji} = \eta(t_j - o_j)o_j(1-o_j)o_i$$

Observations from Δw_{ji}

$$\Delta w_{ji} = \eta(t_j - o_j)o_j(1-o_j)o_i$$

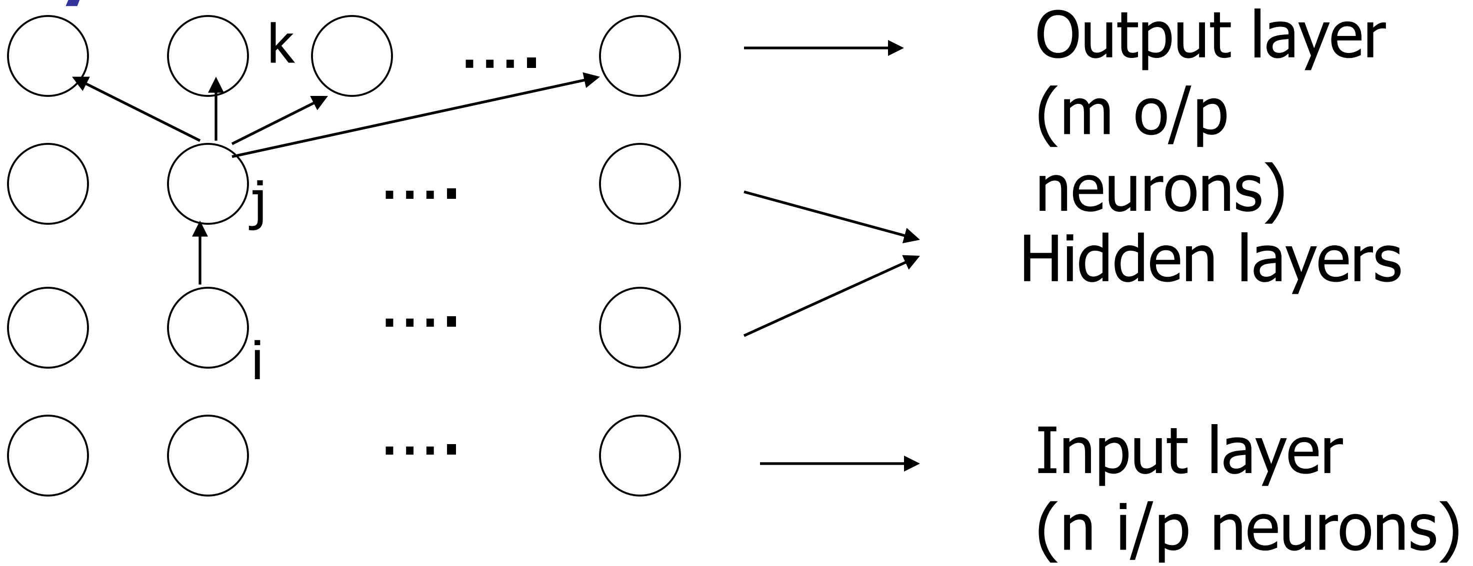
$$\Delta w_{ji} \to 0 \quad \text{if,}$$

$$1.\ o_j \to t_j \quad \text{and/or}$$

$$\left.\begin{array}{l} 2.\ o_j \to 1 \quad \text{and/or} \\ 3.\ o_j \to 0 \quad \text{and/or} \end{array}\right\} \text{Saturation behaviour}$$

$$4.\ o_i \to 0 \qquad \} \text{Credit/Blame assignment}$$

Backpropagation for hidden layers



δ_k is propagated backwards to find value of δ_j

Backpropagation – for hidden layers

$$\Delta w_{ji} = \eta \delta j o_i$$

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j}$$

$$= -\frac{\delta E}{\delta o_j} \times o_j(1-o_j)$$

This recursion can give rise to vanishing and exploding Gradient problem

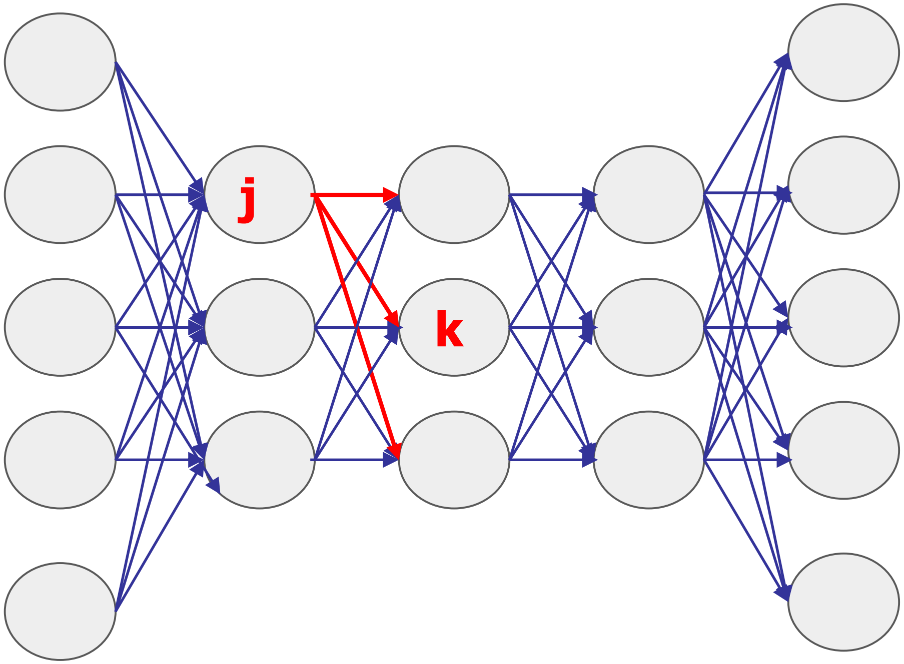
$$= -\sum_{k \in \text{next layer}} \left(\frac{\delta E}{\delta net_k} \times \frac{\delta net_k}{\delta o_j}\right) \times o_j(1-o_j)$$

$$\text{Hence, } \delta_j = -\sum_{k \in \text{next layer}} (-\delta_k \times w_{kj}) \times o_j(1-o_j)$$

$$= \sum_{k \in \text{next layer}} (w_{kj}\delta_k) o_j(1-o_j)$$

phi'(zj) is the derivative of the activation function of neuron j

Back-propagation- for hidden layers: Impact on net input on a neuron



- O_j affects the net input coming to all the neurons in next layer

General Backpropagation Rule

- General weight updating rule:

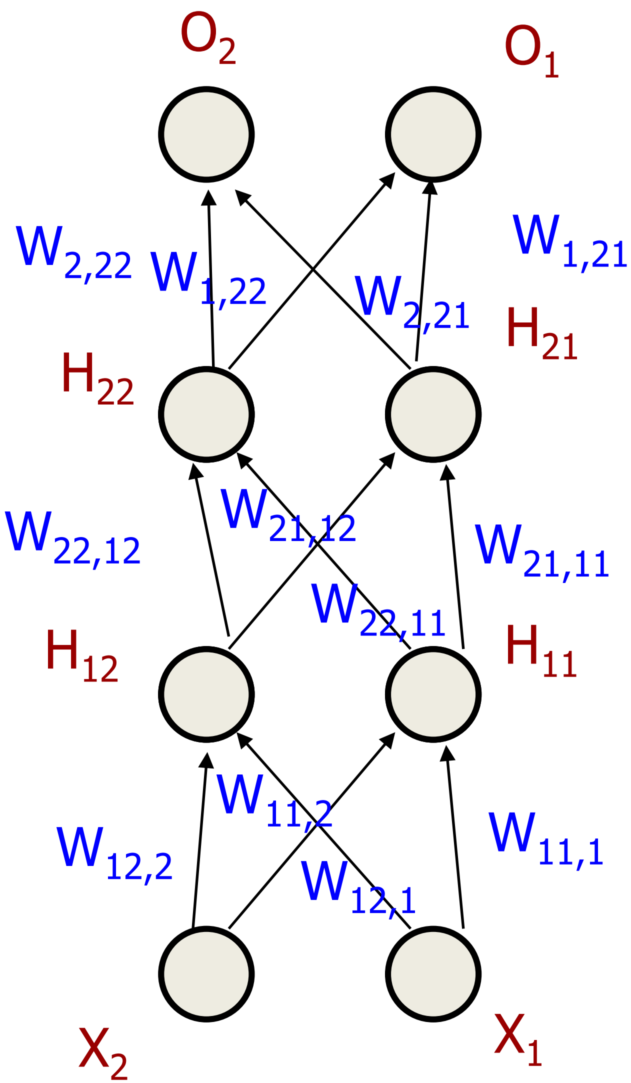
$$\Delta w_{ji} = \eta \delta j o_i$$

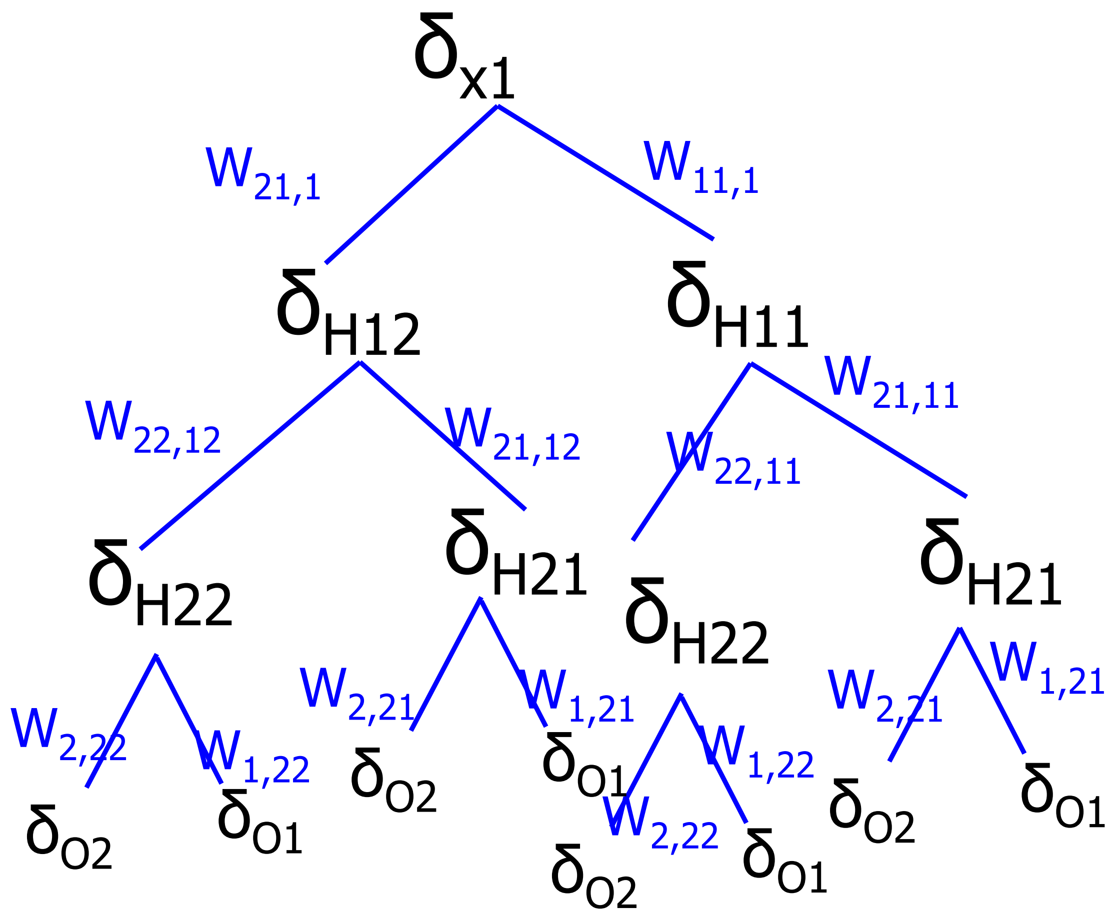
- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) \quad \text{for hidden layers}$$

Vanishing/Exploding Gradient problem





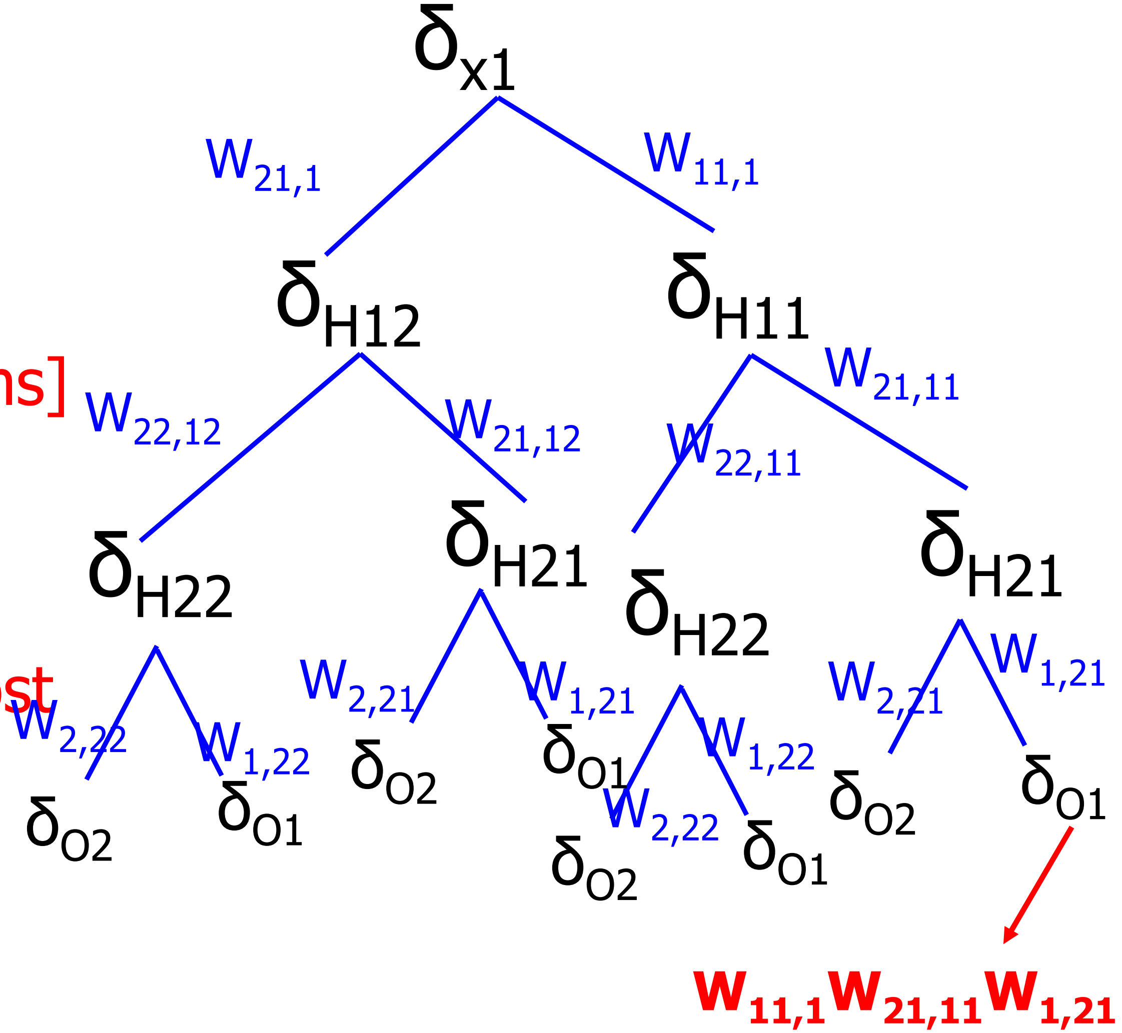
$\delta_{x1}=W_{11,1}\delta_{H11}f'(.)+W_{21,1}\delta_{H1}f'(.)$,
$f'(.)$ is derivative of sigmoid

Vanishing/Exploding Gradient

$\delta_{x1} = W_{11,1}\delta_{H11}f'(.) + W_{21,1}\delta_{H12}f'(.)$ [2 terms]

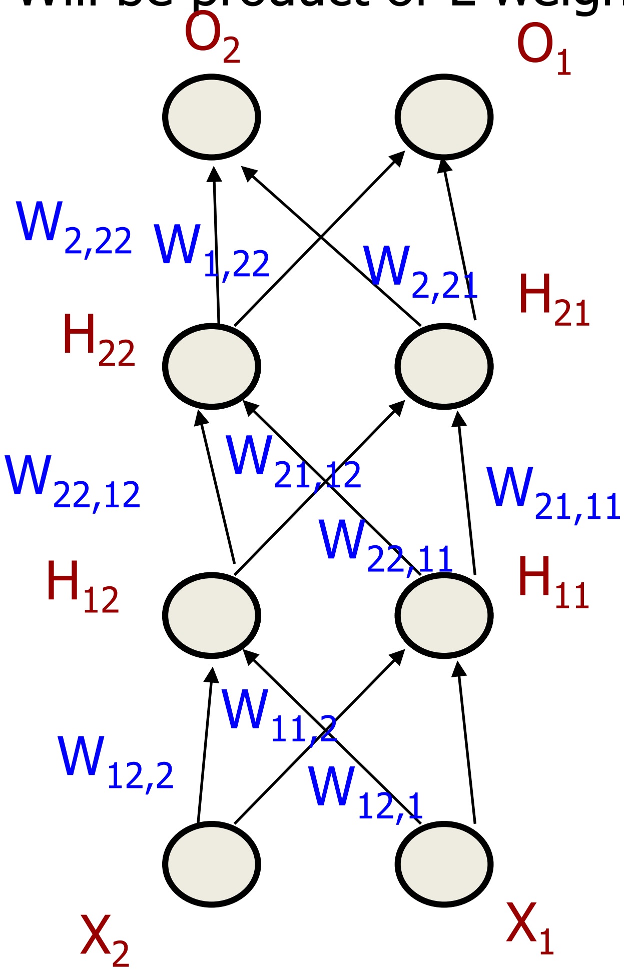
$= W_{11,1}(W_{21,11}\delta_{H21}f'(.) + W_{22,11}\delta_{H22}f'(.)) f'(.) + W_{21,1}(W_{21,12}\delta_{H21}f'(.) + W_{22,12}\delta_{H22}f'(.)) f'(.)$ [4 terms]

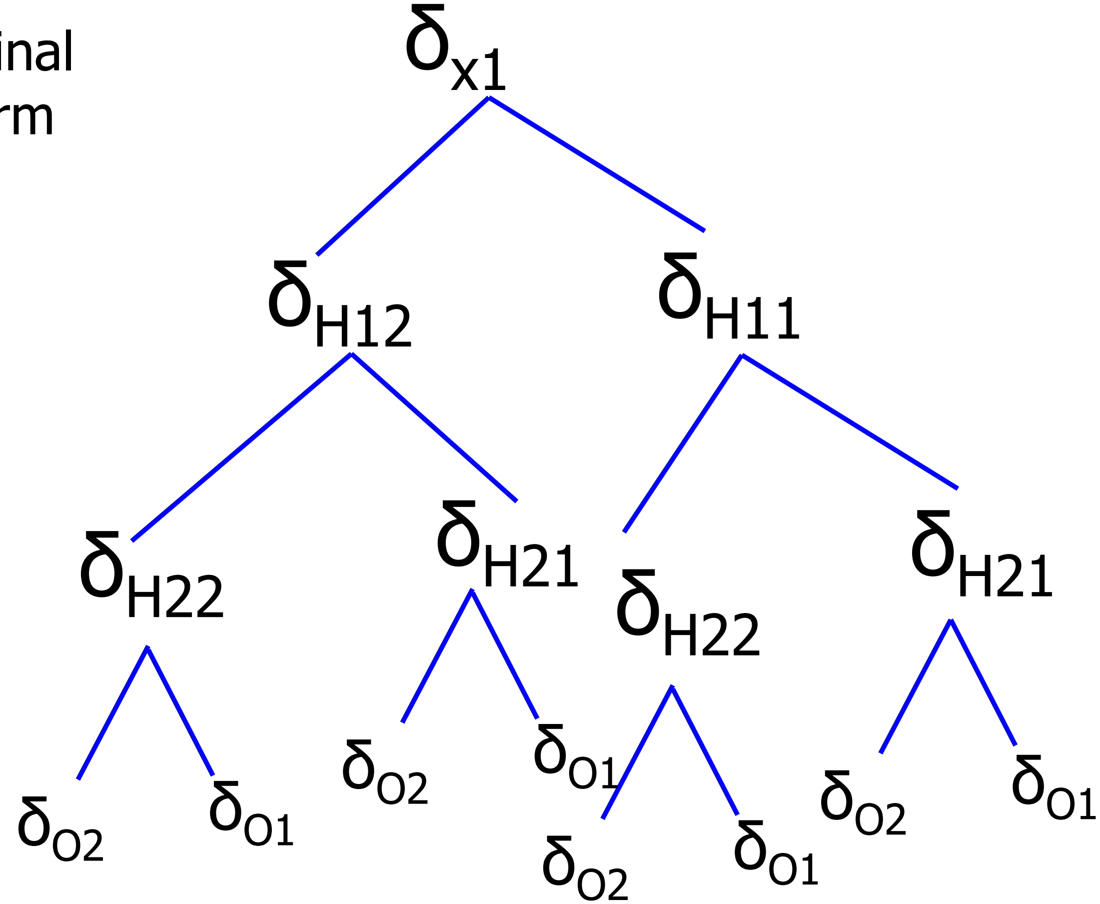
= (4 terms with δ_{o1}) + (4 terms with δ_{o2}; one term shown for the leftmost leaf's weight); also each term has product of derivatives



Vanishing/Exploding Gradient

With '*B*' as branching factor and
'*L*' as number of levels,
There will be B^L terms in the final
Expansion of δ_{x1}. Also each term
Will be product of L weights





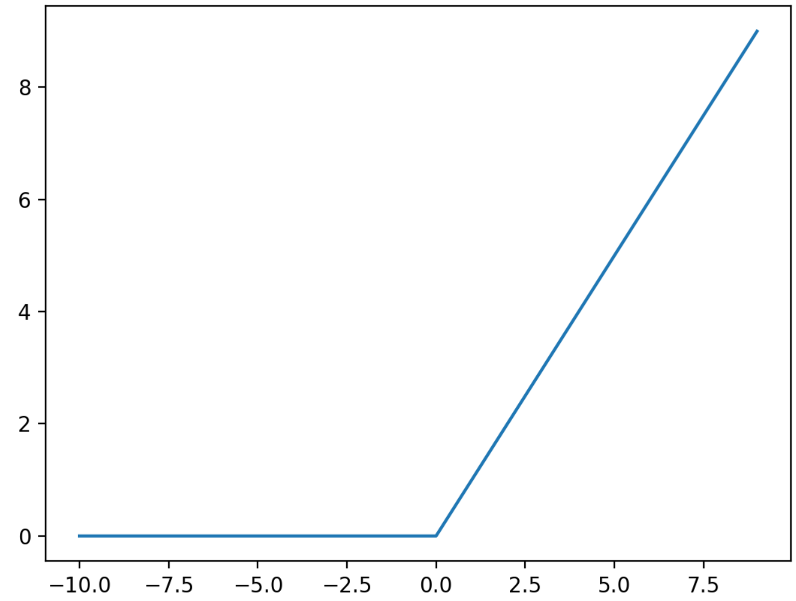
Each term also gets multiplied with
product of derivatives of sigmoid L times.
These products can vanish or explode.

FFNN: Working with RELU

Rectifier Linear Unit

What is RELU

y=relu(x)=max(0,x)



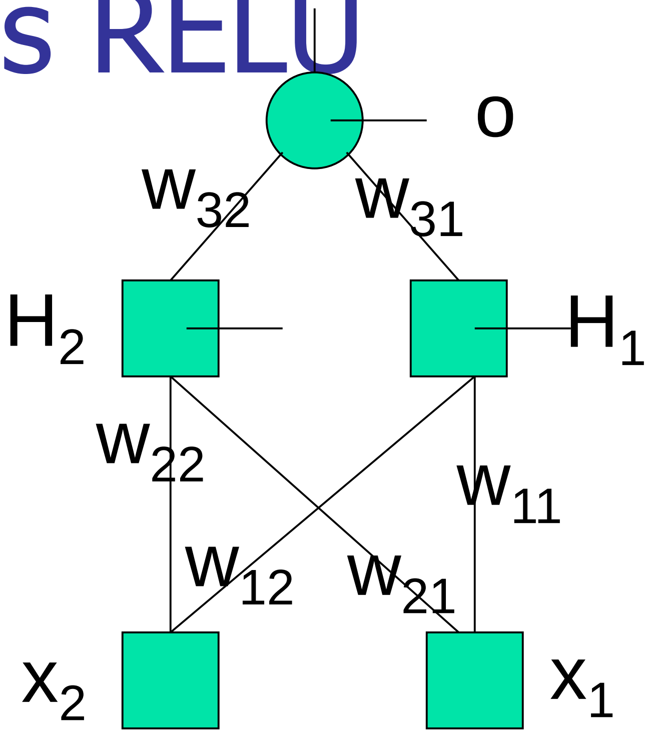
dy/dx

= 0 for x<o

= 1 for x>0

= 0 (forced to be 0 at x=0, though does not exit)

Output sigmod and hidden neurons as RELU



$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}}$$

$$\eta = \text{learning rate, } 0 \leq \eta \leq 1$$

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}}$$

$$net_j = \text{input at the j}^{th} \text{ neuron)}$$

$$\frac{\delta E}{\delta net_j} = -\delta j$$

$$\Delta w_{ji} = \eta \delta j \frac{\delta net_j}{\delta w_{ji}} = \eta \delta j o_i$$

Backpropagation – for outermost layer

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j} (net_j = \text{input at the j}^{th}\text{ layer})$$

$$E = \frac{1}{2}\sum_{p=1}^{m}(t_p - o_p)^2$$

$$\text{Hence, } \delta j = -(-(t_j - o_j)o_j(1-o_j))$$

$$\Delta w_{ji} = \eta(t_j - o_j)o_j(1-o_j)o_i$$

Backpropagation – for hidden layers

$$\Delta w_{ji} = \eta \delta j o_i$$

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j}$$

$$= -\frac{\delta E}{\delta o_j} \times (1\, or\ 0)$$

used neuron depends on netk.

This recursion can give rise to vanishing and exploding Gradient problem

$$= -\sum_{k \in \text{next layer}} \left(\frac{\delta E}{\delta net_k} \times \frac{\delta net_k}{\delta o_j}\right) \times (1\, or\ 0)$$

$$\text{Hence, } \delta_j = -\sum_{k \in \text{next layer}} (-\delta_k \times w_{kj}) \times (1\, or\ 0)$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k)\ or\ 0$$

Backpropagation Rule for weight change with RELU, Sigmoid and TSS

$$\Delta w_{ji} = \eta \delta j o_i$$

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) \; or \; 0 \quad \text{for hidden layers}$$

Softmax, Cross Entropy and RELU

Cross Entropy Function

$$H(P,Q) = -\sum_{x} P(x)\log_2 Q(x)$$

P is target distribution, *Q* is observed distribution

e.g., Positive, Negative, Neutral Sentiment

x: input sentence: *The movie was excellent*

P(x): <1,0,0>, Q(x): <0.9,0.02,0.08>, (say)

H(P,Q)=-log0.9=log(10/9)

Deriving weight change rules

Cross Entropy Softmax combination

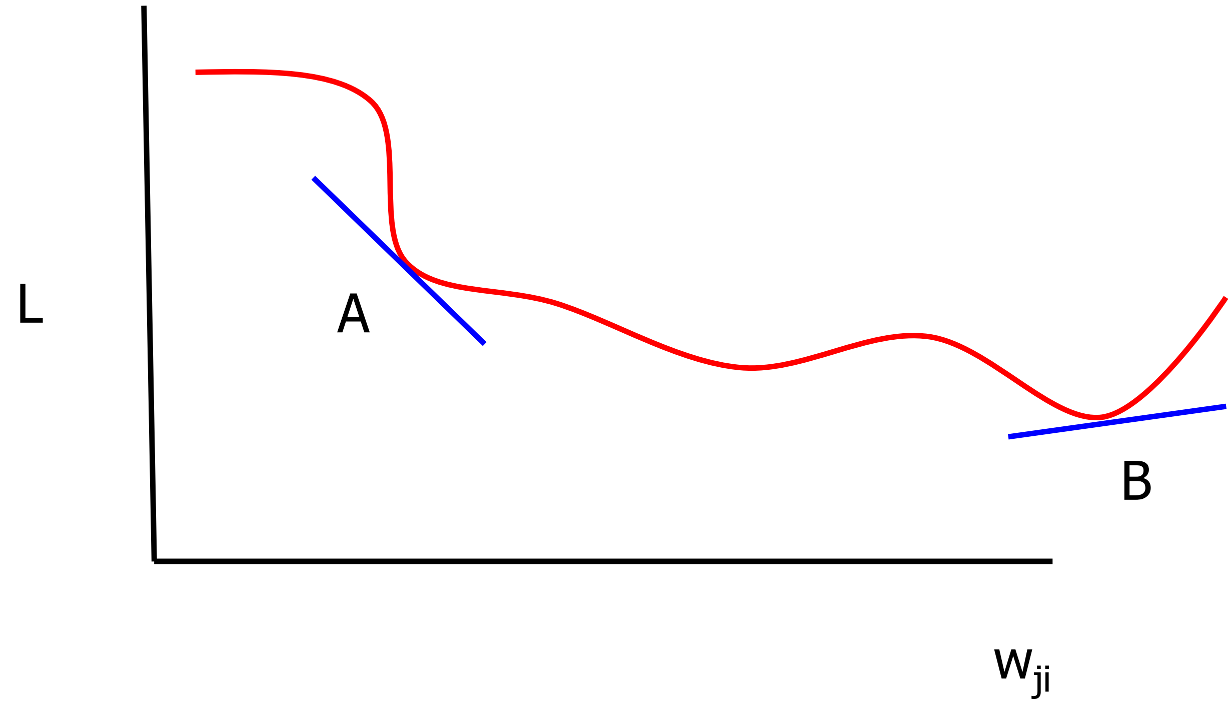
A very ubiquitous combination in neural combination

Foundation: Gradient descent

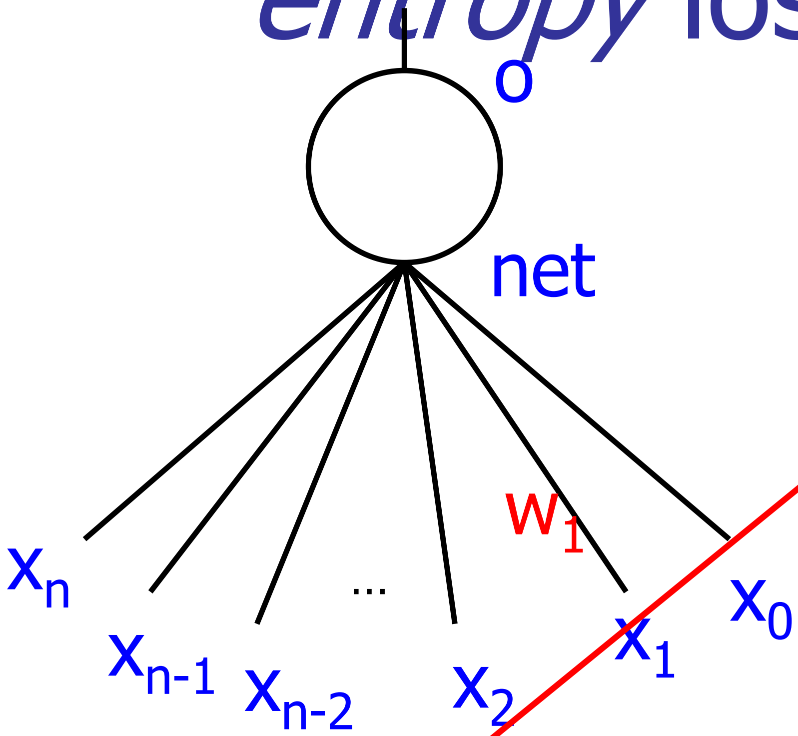
Change is weight $\Delta w_{ji} = -\eta \delta L / \delta w_{ji}$

η= learning rate, L=loss, w_{ji}= weight of connection from the i^{th} neuron to j^{th}

At A, *$\delta L / \delta w_{ji}$ is negative, so Δw_{ji} is positive. At B, $\delta L / \delta w_{ji}$ Is positive, so so Δw_{ji} is negative. L *always decreases. Greedy algo.*



Single neuron: *sigmoid+cross entropy* loss



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial o} \cdot \frac{\partial o}{\partial net} \cdot \frac{\partial net}{\partial w_1}$$

$$L = -t \log o - (1-t)\log(1-o)$$

$$\Rightarrow \frac{\partial L}{\partial o} = -\frac{t}{o} + \frac{1-t}{1-o} = -\frac{t-o}{o(1-o)} \quad (1)$$

$$o = \frac{1}{1+e^{-net}} \; (sigmoid) \Rightarrow \frac{\partial o}{\partial net} = o(1-o) \; (2)$$

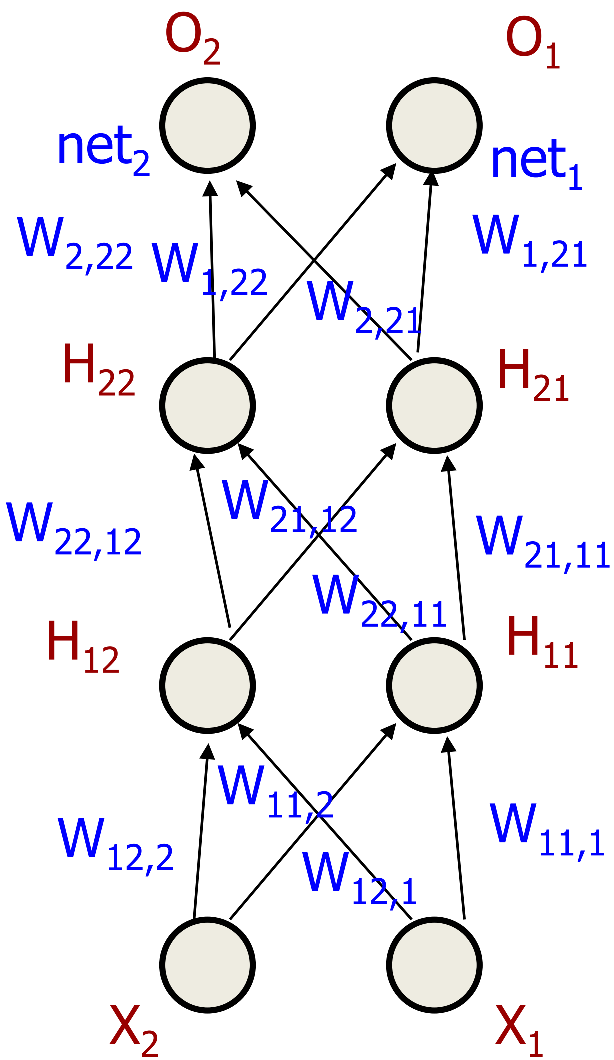
$$net = \sum_{i=0}^{n} w_i x_i \;\; \Rightarrow \frac{\partial net}{\partial w_1} = x_1 \;\; (3)$$

$$\Rightarrow \Delta w_1 = \eta \frac{\partial L}{\partial w_1} = \eta (t-o) x_1$$

The same result comes for SoftMax with cross entropy

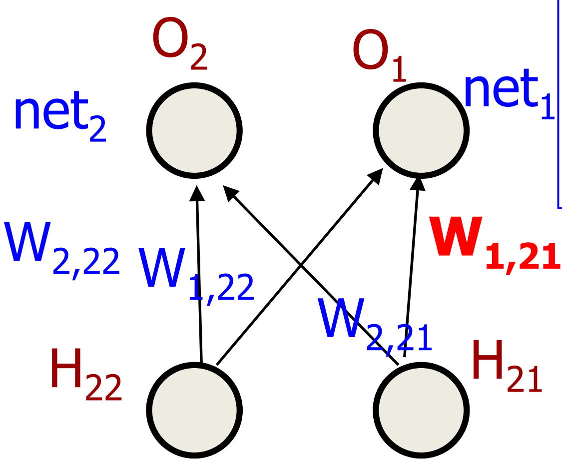
$$\Delta w_1 = \eta(t\text{-}o)x_1$$

FFNN with O_1-O_2 softmax, all hidden neurons RELU, Cross Entropy Loss



We will apply the $\Delta w_{ji}=\eta\delta_j o_i$ rule

General Weight Change Equation



$$\Delta w_{1,21} = \eta \delta_{o_1} h_{21}$$

$$\delta_{o_1} = -\frac{\delta E}{\delta net_1}$$

$$E = -t_2 \log o_2 - t_1 \log o_1$$

$$\frac{\partial E}{\partial net_1} = \frac{\partial E}{\partial o_1} \cdot \frac{\partial o_1}{\partial net_1} + \frac{\partial E}{\partial o_2} \cdot \frac{\partial o_2}{\partial net_1}$$

$$= -\frac{t_1}{o_1} o_1 (1 - o_1) + (-\frac{t_2}{o_2})(-o_1 o_2)$$

$$= -t_1 (1 - o_1) + t_2 o_1$$

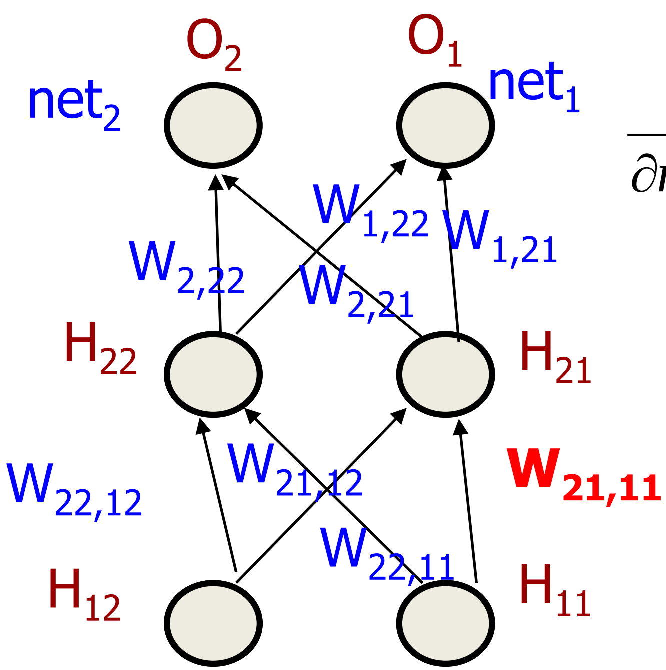
$$= -t_1 o_2 + t_2 o_1 = -(t_1 - o_1)$$

$$\Rightarrow \delta_{O_1} = (t_1 - o_1)$$

$$Similarly, \delta_{O_2} = (t_2 - o_2)$$

$$\boxed{\Delta W_{1,21} = \eta (t_1 - o_1) h_{21}}$$

Weight Change for Hidden Layer, $W_{21,11}$



$$\Delta w_{21,11} = -\eta \frac{\partial E}{\partial w_{21,11}} = \eta \delta_{H_{21}} h_{11}$$

$$\delta_{H_{21}} = -\frac{\partial E}{\partial net_{H_{21}}}$$

$$\frac{\partial E}{\partial net_{H_{21}}} = \frac{\partial E}{\partial h_{21}} . \frac{\partial h_{21}}{\partial net_{H_{21}}}; h_{21} = output(H_{21})$$

$$= \frac{\partial E}{\partial h_{21}} .r'(H_{21}); \quad r' = derivative\_RELU(H_{21})$$

$$\frac{\partial E}{\partial h_{21}} = \frac{\partial E}{\partial net_1} . \frac{\partial net_1}{\partial h_{21}} + \frac{\partial E}{\partial net_2} . \frac{\partial net_2}{\partial h_{21}}$$

$$= (-\delta_{o_1}).W_{1,21} + (-\delta_{o2}).W_{2,21}$$

$$\Rightarrow \delta_{H_{21}} = (\delta_{o_1}.W_{1,21} + \delta_{o2}.W_{2,21}).r'(H_{21})$$

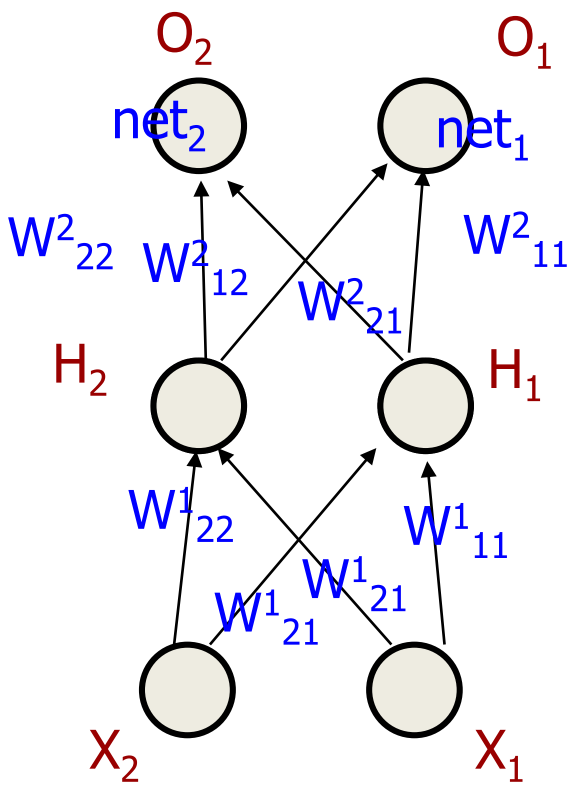
$$= backpropagated\_delta.RELU\_derivative$$

$$\boxed{\Delta W_{21,11} = \eta[(t_2 - o_2)W_{2,21} + (t_1 - o_1)W_{1,21}].r'(H_{21}).h_{11}}$$

Example

There is a pure feedforward network 2-2-2 (2 input, 2 hidden and 2 output neurons). Input neurons are called X_1 and X_2 (right to left when drawn on paper, X_1 to the right of X_2). Similarly hidden neurons are H_1 and H_2 (right to left) and output neurons are O_1 and O_2 (right to left). H_1 and H_2 are RELU neurons. O_1 and O_2 form a softmax layer.

Remember: weight change rules



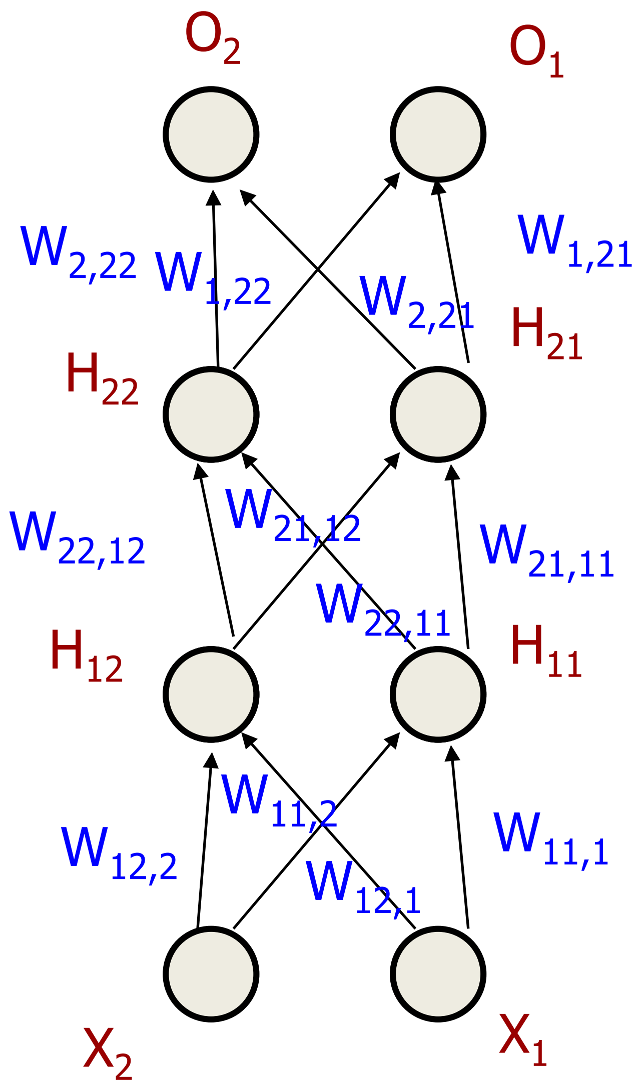
$$E = -t_2 \log o_2 - t_1 \log o_1$$

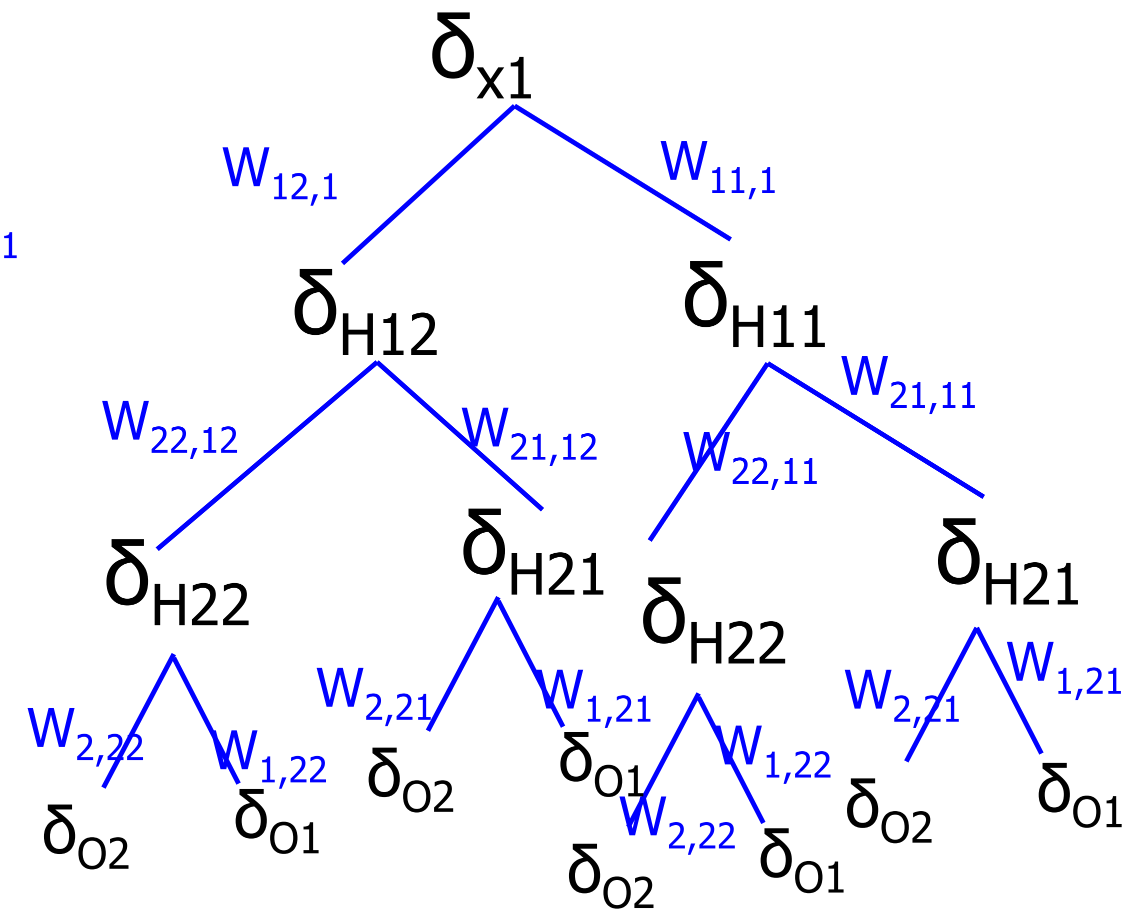
$$\boxed{\Delta W^2_{11} = \eta(t_1 - o_1)h_1}$$

$$\boxed{\mathbf{\Delta W^1_{11} = \eta[(t_2 - o_2)W^2_{21} + (t_1 - o_1)W^1_{11}].r'(H_1).X_1}}$$

Why is RELU a solution for vanishing or exploding gradient?

Vanishing/Exploding Gradient





$$\delta_{x1} = W_{11,1}\delta_{H11} + W_{12,1}\delta_{H12}$$

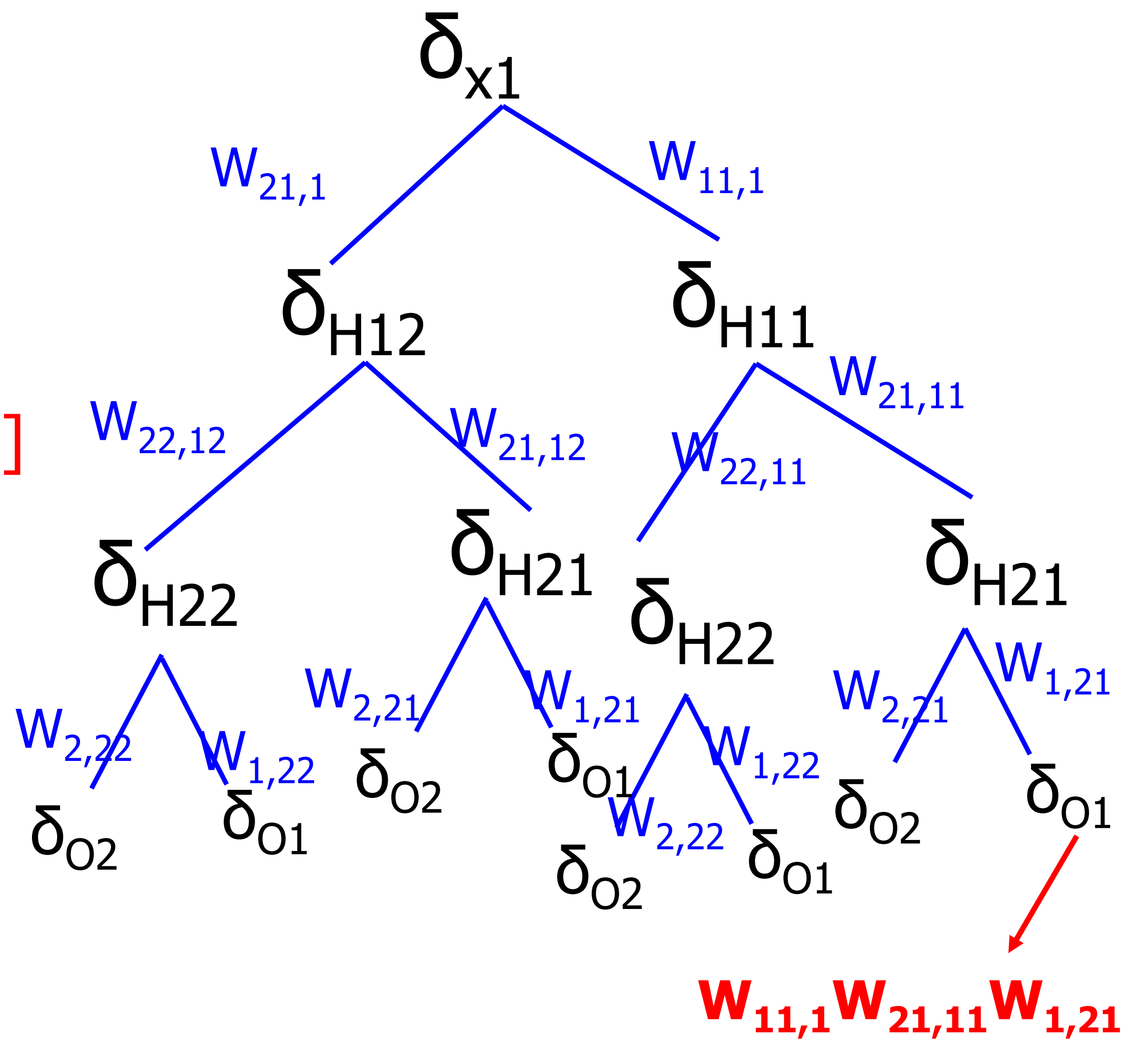
Vanishing/Exploding Gradient

$\delta_{x1}=W_{11,1}\delta_{H11}+W_{21,1}\delta_{H12}$ [2 terms]

$=W_{11,1}(W_{21,11}\delta_{H21}+W_{22,11}\delta_{H22}).r'(H_{11})+W_{21,1}(W_{21,12}\delta_{H21}+W_{22,12}\delta_{H22}).\ r'(H_{12})$ [4 terms]

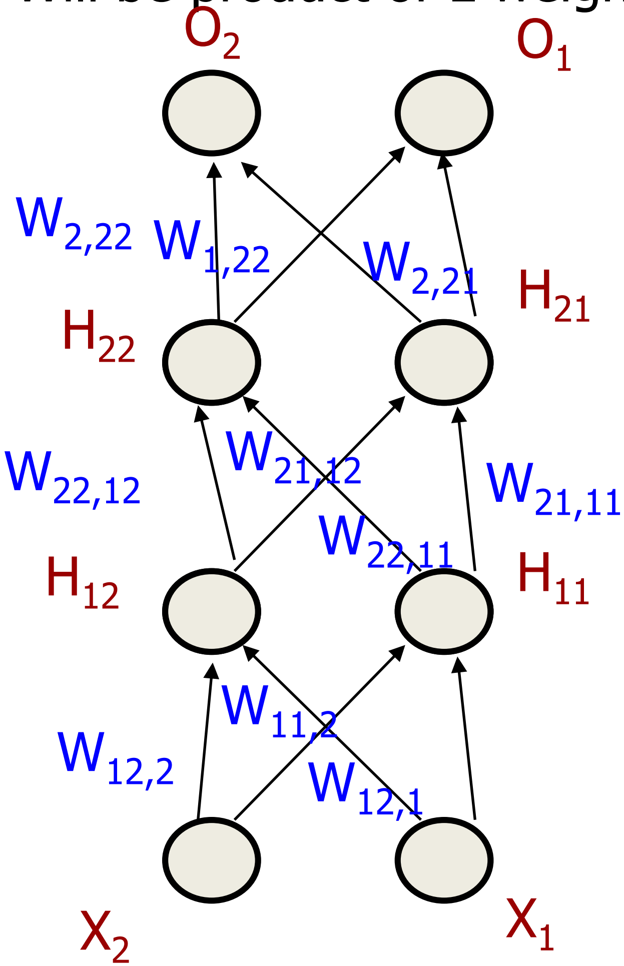
= (4 terms involving δ_{o1}) + (4 terms involving δ_{o2})

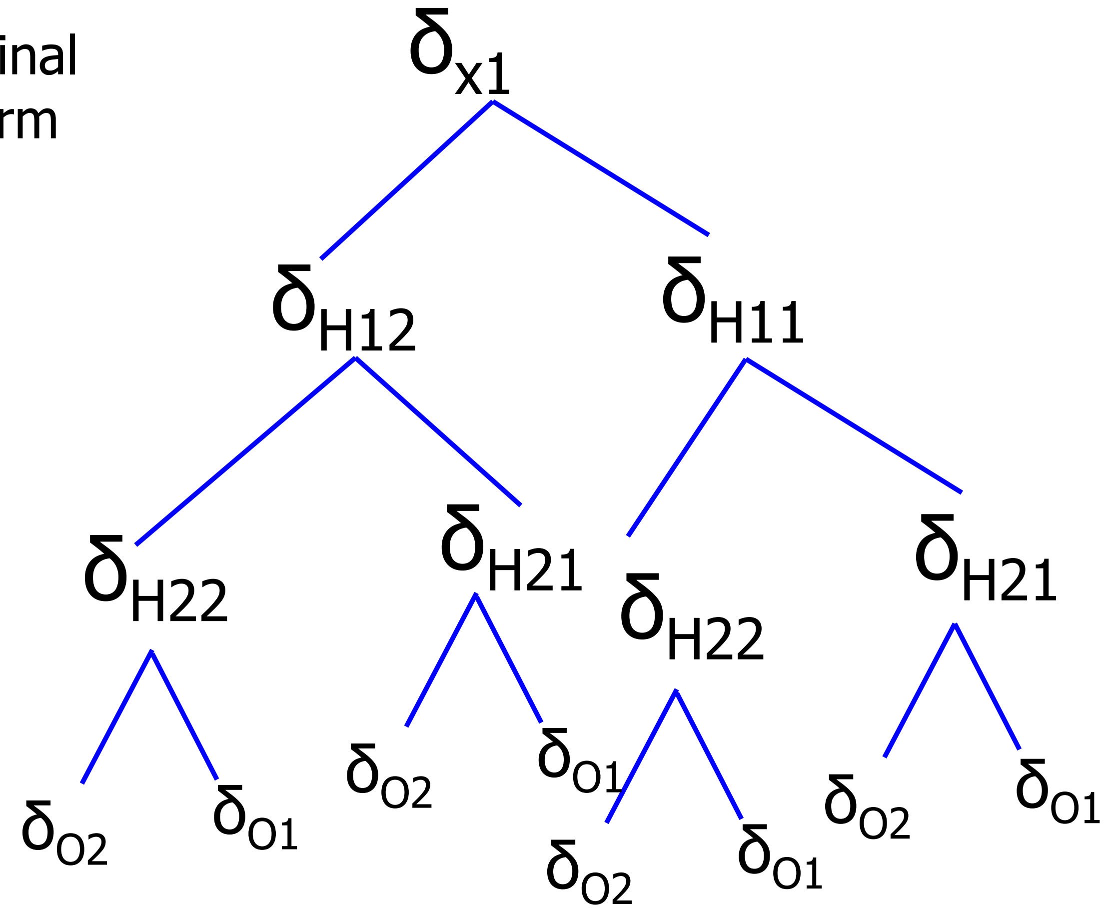
δs get multiplied by derivatives of RELU which are 1 or 0; hence δs from the output layer pass as such or as 0



Vanishing/Exploding Gradient

With '*B*' as branching factor and
'*L*' as number of levels,
There will be B^L terms in the final
Expansion of δ_{x1}. Also each term
Will be product of L weights





How can gradients explode

- Station derivatives multiply
- If <0, progressive attenuation of product
- Now the sigmoid function can be in the form of $y=K[1/(1+e^{-x})]$
- Derivative= $K.y.(1-y)$
- If K is more than 1, the product of gradients can become larger and larger, leading to explosion of gradient
- K needs to be >1, to avoid saturation of neurons

Can happen for *tanh* too

- Tanh: $y=[(e^x-e^{-x})/(e^x+e^{-x})]$
- Derivative= (1-y)(1+y)
- If we take a neuron with *K.tanh*, we can again have explosion of gradient if K>1
- Why *K* needs to be >1?
- To take care of situations where #inputs and individual components of input are large
- This is to avoid saturation of the neuron