CS626: Speech, Natural Language Processing and the Web

Sigmod, Softmax, FFNN, BP

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

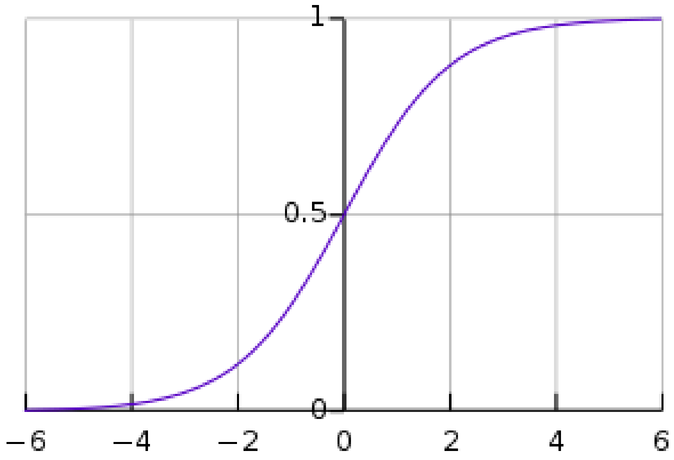
Week 8 of 12th September, 2022

2-class: Sigmoid or Logit function

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

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

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

multiclass: SOFTMAX

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

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

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

(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	**Positive**	**Negative**	**Neutral**
Sent (a)	1 *(P_{max} from softmax)*	0	0
Sentence (b)	0	1 *(P_{max} from softmax)*	0
Sentence (C)	0	0`	1 *(P_{max} 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

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

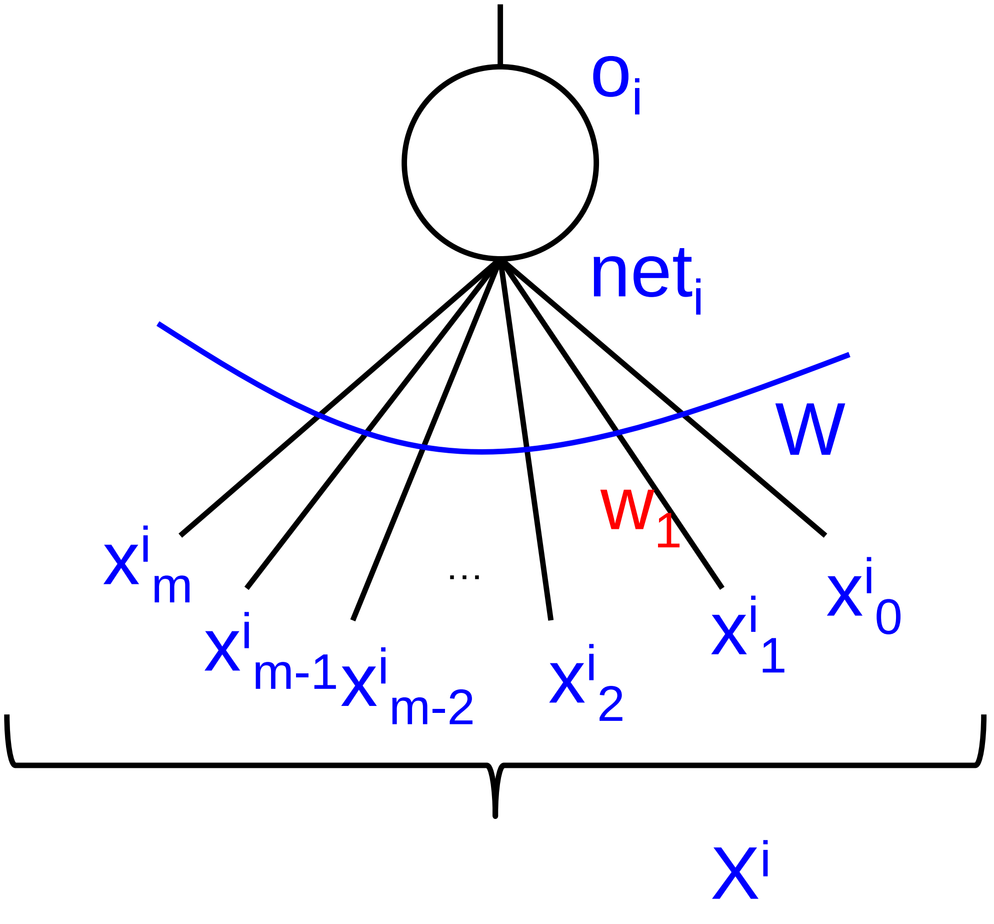
P is target distribution
Q is observed distribution

How to minimize loss

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

Sigmoid and Softmax neurons

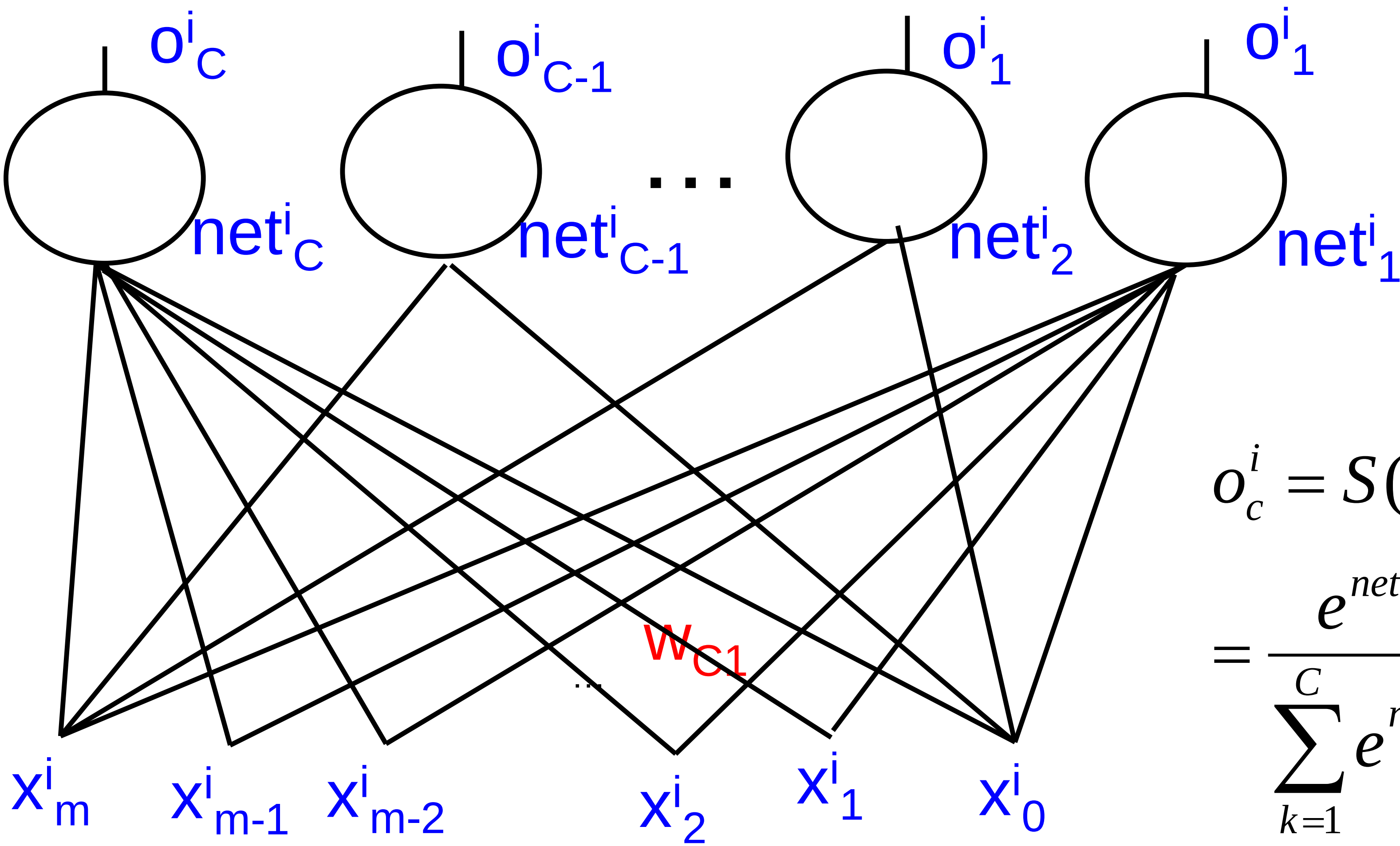
Sigmoid neuron



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

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

Softmax Neuron



$$o^i_c = S(NET^i)_c$$

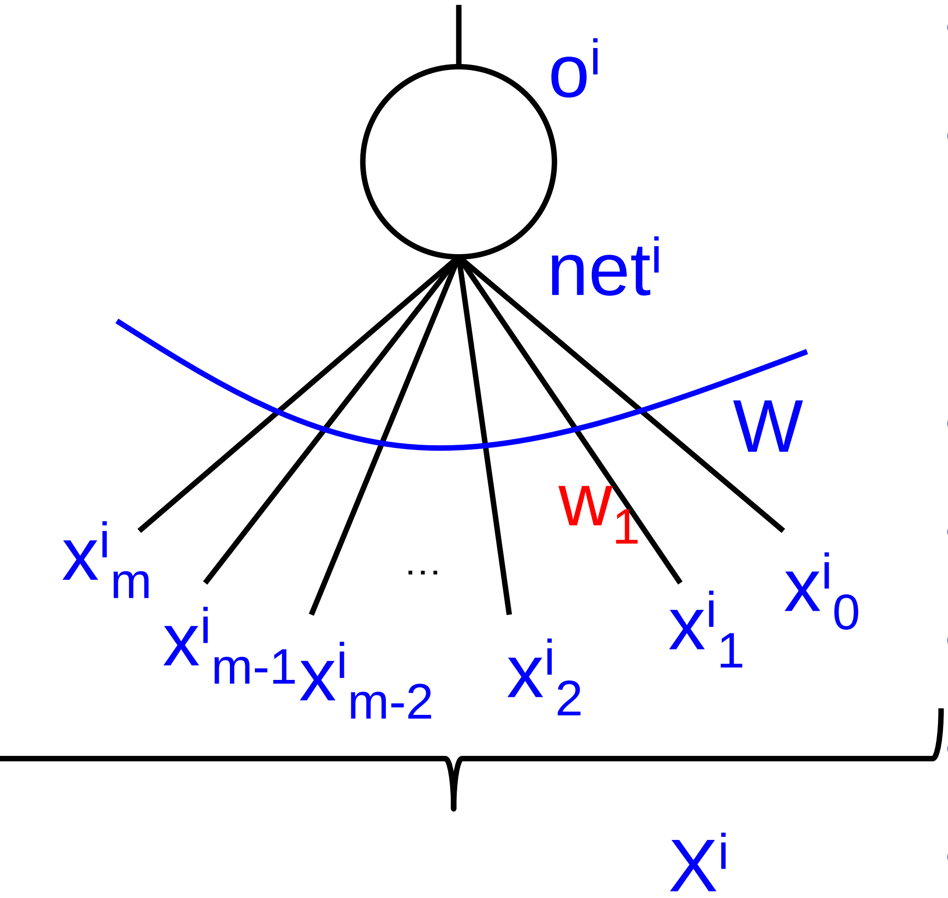
$$= \frac{e^{net^i_c}}{\sum_{k=1}^{C} e^{net^i_k}}$$

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

Notation

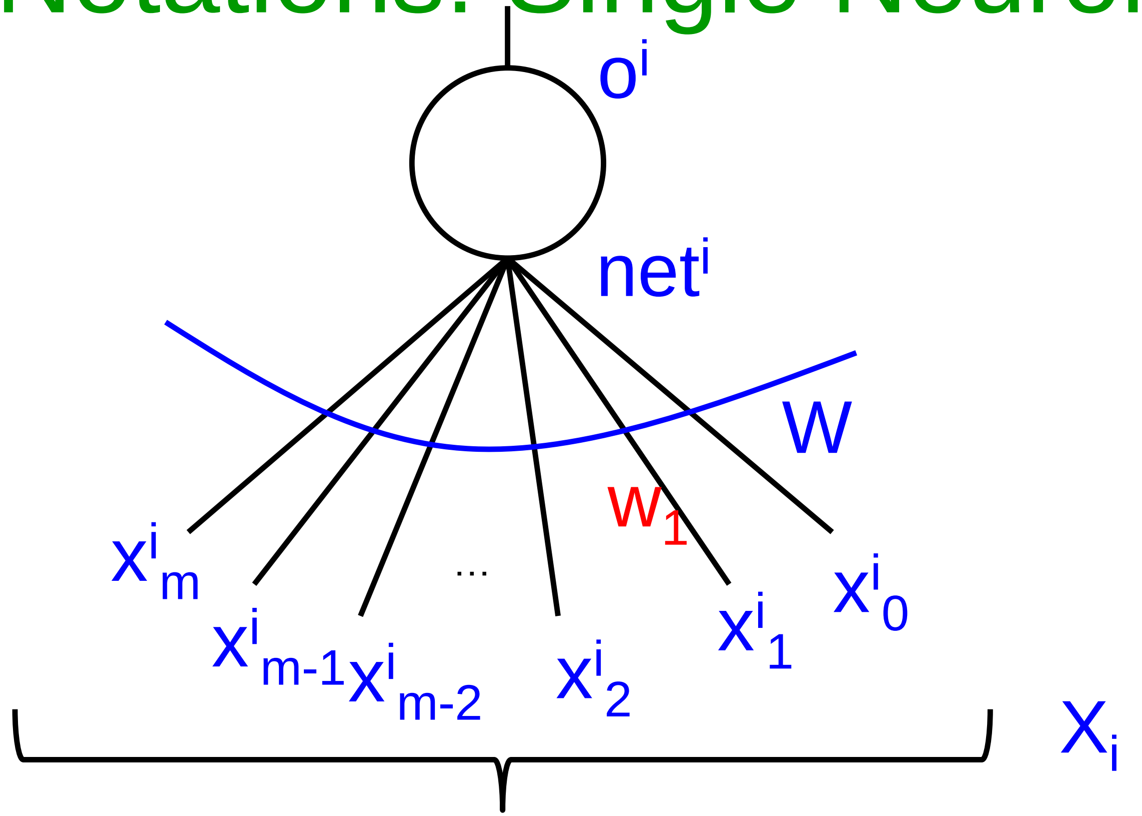
- $i=1..N$
- N i-o pairs, i runs over the training data
- $j=0\ldots m$, m components in the input vector, j runs over the input dimension (also weight vector dimension)
- $k=1\ldots C$, C classes (C components in the output vector)

Fix Notations: Single Neuron (1/2)



- Capital letter for vectors
- Small letter for scalars (therefore for vector components)
- X^i: ith 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)



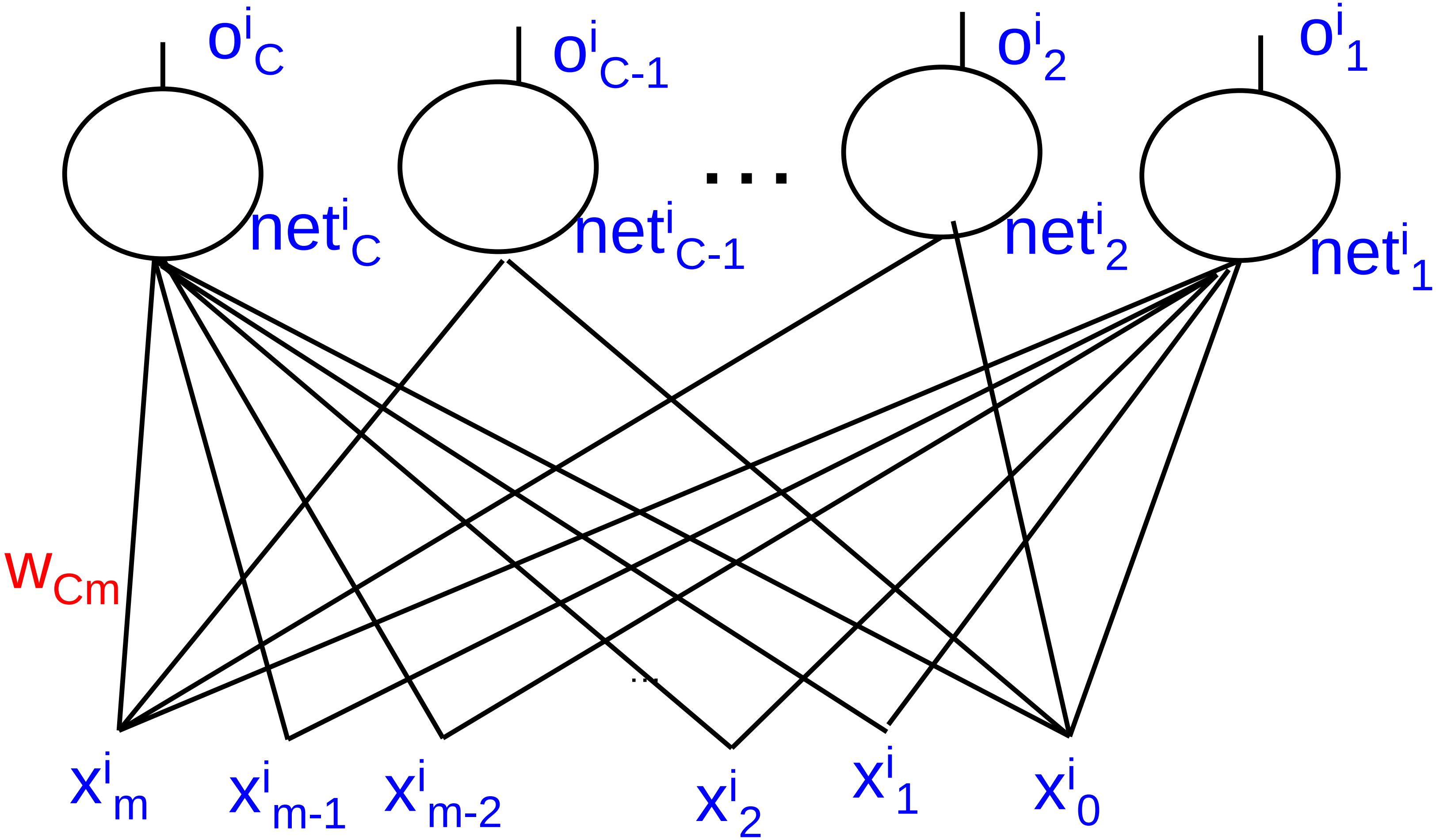
W and each X^i has *m* components

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

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

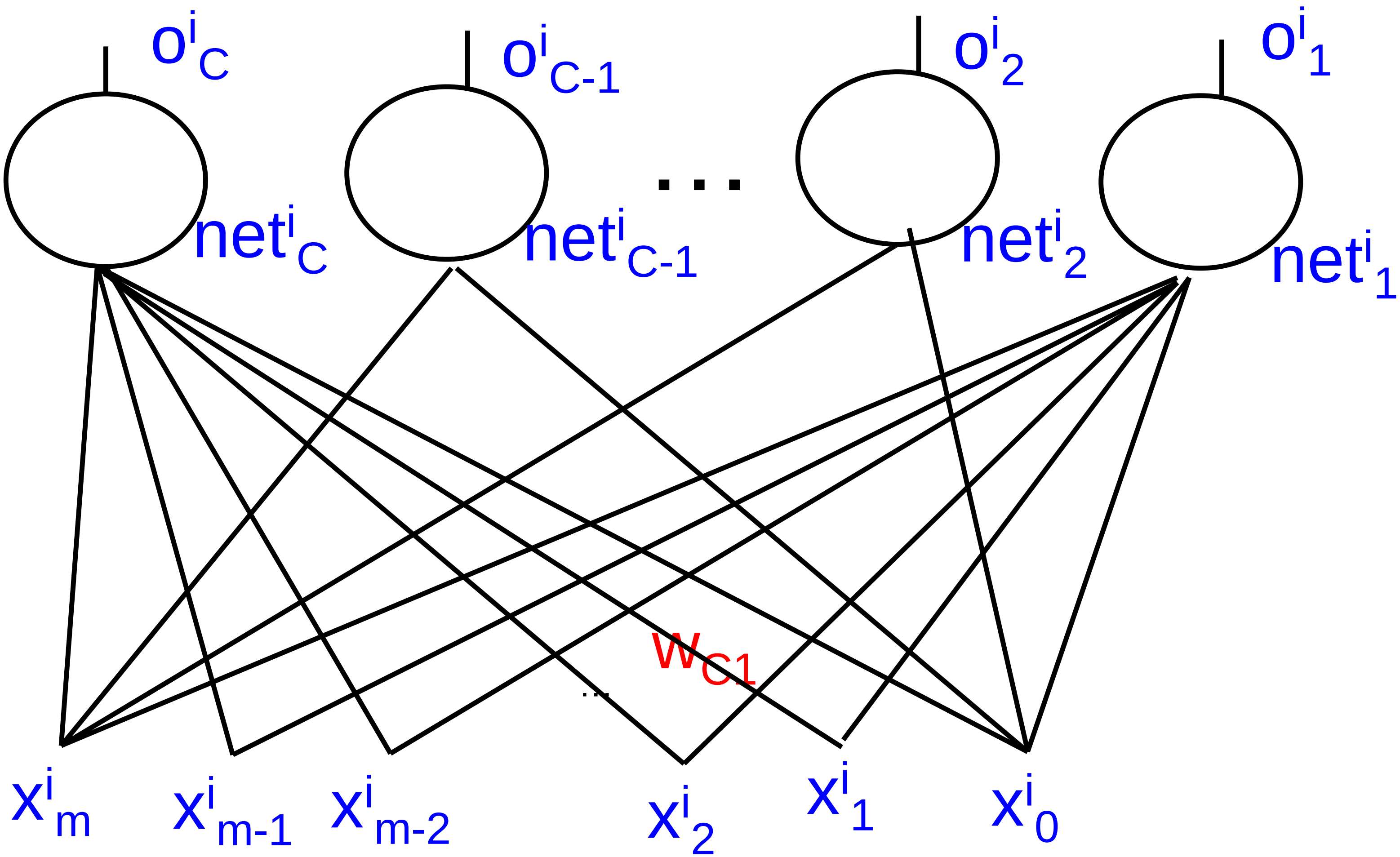
Upper suffix *i* indicates i^{th} input

Fixing Notations: Multiple neurons in o/p layer



Now, O^i and NET^i are vectors for i^{th} input W_k is the weight vector for k^{th} output neuron, $k=1..C$

Fixing Notations



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

Maximum Likelihood and Cross Entropy Loss

Cross Entropy Function

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

P is target distribution

Q is observed distribution

X varies over i-o pairs, i.e., training data instances

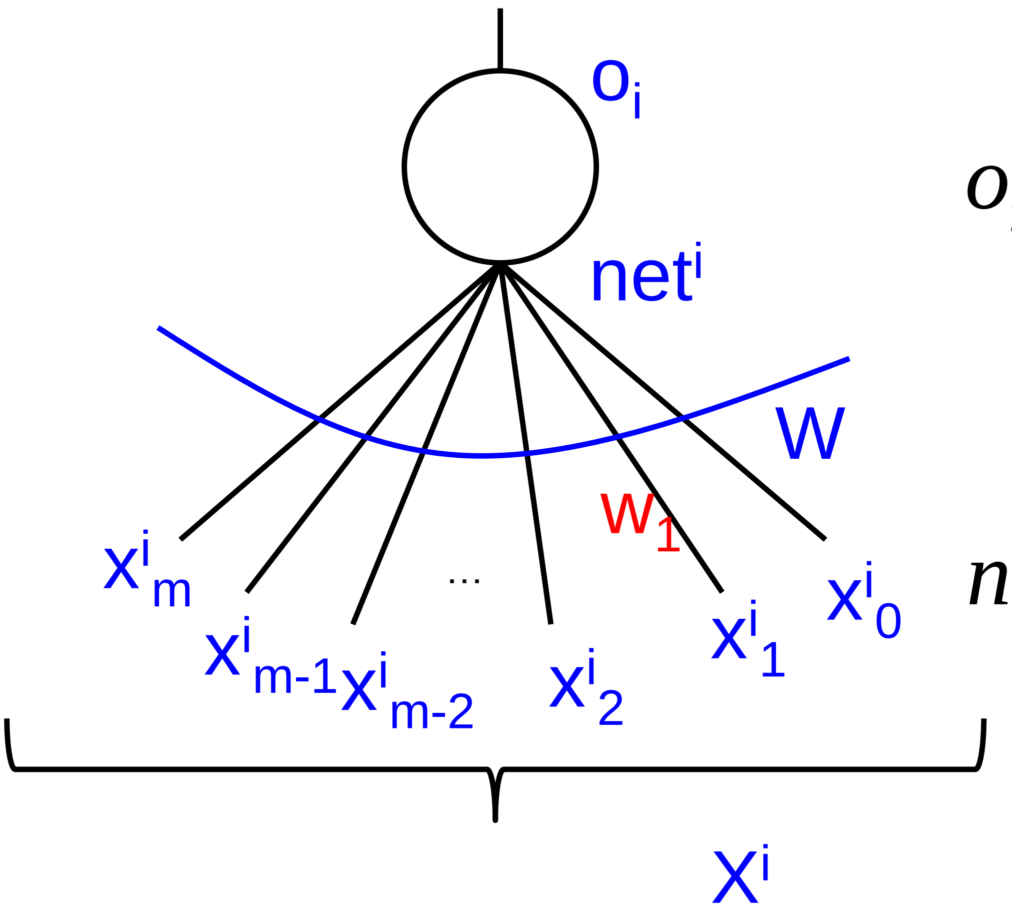
Fixing concepts

- The random variable is the class value of the input
- So we are interested in the probability

  $P(O^i|X^i)$, where O^i is the output vector given the input vector X^i
- Each component o^i_c of O^i is the probability of X_i belonging to the class c ($c=1\ldots C$)
- Notice that C components are redundant, since probability(class-c)=

  1-Σprobability(class≠c)
- So in case of 2-class, one sigmoid neuron

Sigmoid neuron



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

$$net^i = W.X_i = \sum_{j=0}^{m} w_j x^i_j$$

Interpreting o_i

- o^i value is between 0 and 1
- Interpreted as probability
- 2-class situation, o^i value is looked upon as probability of class being 1
- That is, $P(Class=1 \text{ for } i^{th} \text{ input})$

  $= o^i = 1/(1+e^{-net_i})$
- Each training data instance is labeled as 1 or 0
- Target value $t^i=1/0$, for i^{th} input

Likelihood *L* of observation

$$For\ N\ no.of\ i-o\ pairs$$

$$L=\prod_{i=1}^{N}(o^i)^{t^i}(1-o^i)^{(1-t^i)},\ t^i=1/0$$

$$\log likelihood,\ LL=\sum_{i=1}^{N}\ t^i\log o^i+(1-t^i)\log(1-o^i)$$

$$-LL=-\sum_{i=1}^{N}\ [t^i\log o^i+(1-t^i)\log(1-o^i)]$$

Maximize likelihood=Minimize cross entropy

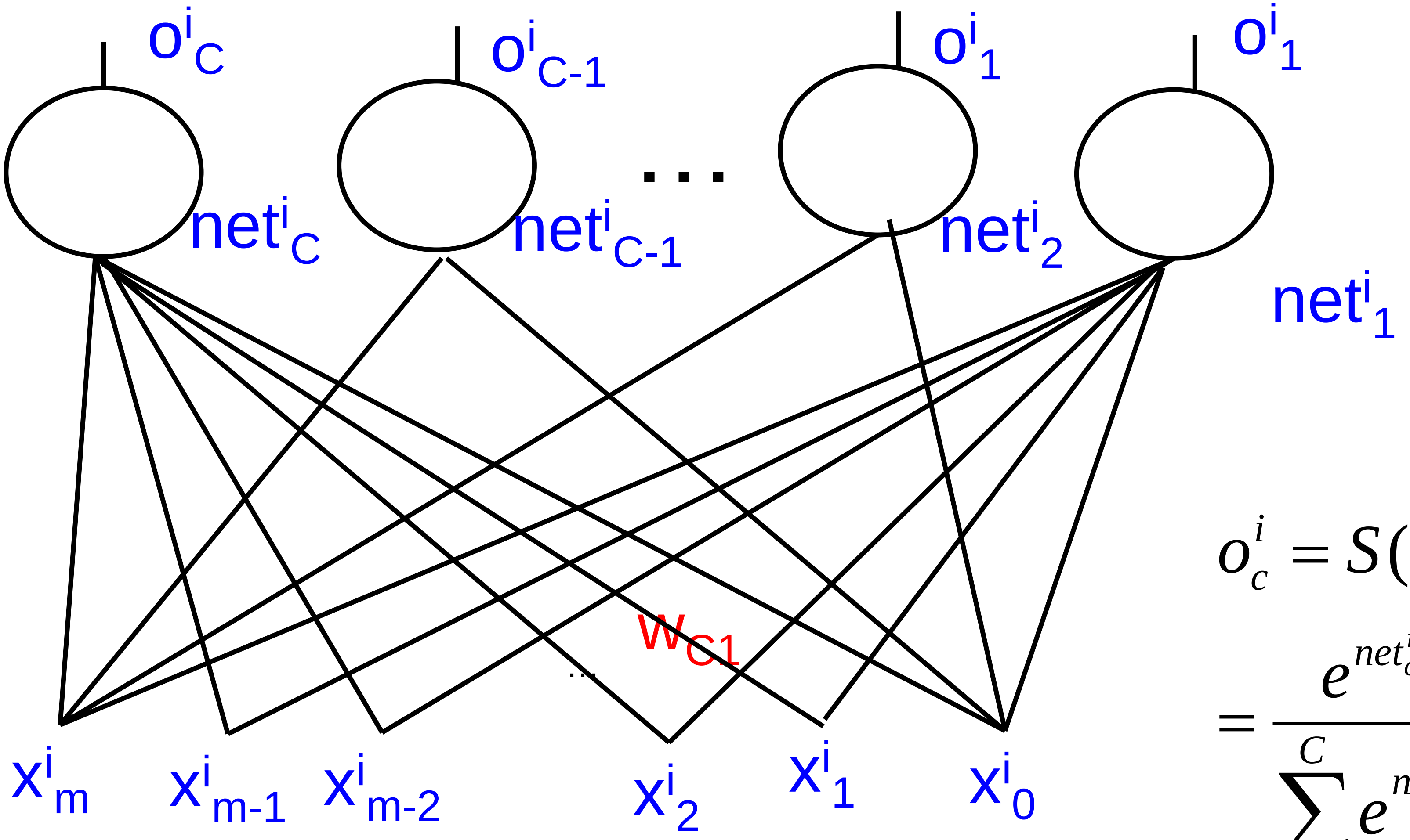
- *-LL* is called the cross entropy
- Regarded as loss or error
- We give this the notation E
- Minimizing cross entropy brings o^i close to t^i (Why?)
- Established: equivalence between maximization of likelihood observation and minimization of cross entropy loss

Generalizing 2-class to multiclass: SOFTMAX

$$o_c^i = S(NET^i)_c = \frac{e^{net_c^i}}{\sum_{k=1}^{C} e^{net_k^i}},$$

- 2-class → multi-class (C classes)
- Sigmoid → softmax
- *i*th input, *c*th class (small c), *k* varies over classes

Softmax Neuron



$$o^i_c = S(\bar{NET}_i)_c = \frac{e^{net^i_c}}{\sum_{k=1}^{C} e^{net^i_k}},$$

Target Vector, T^i: $<t^i_C\ t^i_{C-1} \ldots t^i_2\ t^i_1>$, i → for i^{th} input.
Only one of these C componets is 1, rest are 0.

Compare and contrast Sigmoid and Softmax

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

$$soft\max : o_c^i = \frac{e^{net_c^i}}{\sum_{k=1}^{C} e^{net_k^i}},$$

ith input, *c*th class (small c), *k* varies over classes 1 to *C*

Interpreting o^i_c

- o^i_c value is between 0 and 1
- Interpreted as probability
- Multi-class situation
- o^i_c value is the probability of the class being *'c'* for the i^{th} input

- That is,

$$P(Class\ of\ i^{th}\ input=c)=o^i_c$$

Likelihood L of observations in case of softmax

$$For\ N\ no.of\ i-o\ pairs$$

$$L=\prod_{i=1}^{N}\prod_{k=1}^{C}(o_k^i)^{t_k^i},\ t_k^i=1/0$$

$$For\ a\ pattern\ i,\ only\ one\ of\ t_k^i s\ is\ 1,\ rest\ are\ 0$$

$$\log likelihood,\ LL=\sum_{i=1}^{N}\sum_{k=1}^{C}t_k^i\log o_k^i$$

$$-LL=-\sum_{i=1}^{N}\sum_{k=1}^{C}t_k^i\log o_k^i$$

For softmax also Maximize likelihood=Minimize cross entropy

- *-LL* is called the cross entropy
- Regarded as loss or error
- Given the notation E

- Established again: equivalence between maximization of likelihood of observation and minimization of cross entropy loss

Derivatives

Derivative of sigmoid

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

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

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

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

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

$$\ln o^i_c = net^i_c - \ln\left(\sum_{k=1}^{C} e^{net^i_k}\right)$$

$$\frac{1}{o^i_c}\frac{\partial o^i_c}{\partial net^i_{c'}} = 0 - \frac{1}{\sum_{k=1}^{C} e^{net^i_k}}.e^{net^i_{c'}} = -o^i_{c'}$$

$$\Rightarrow \frac{\partial O^i_k}{\partial net^i_{c'}} = -o^i_c o^i_{c'}$$

Finding weight change rule

Foundation: Gradient descent

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

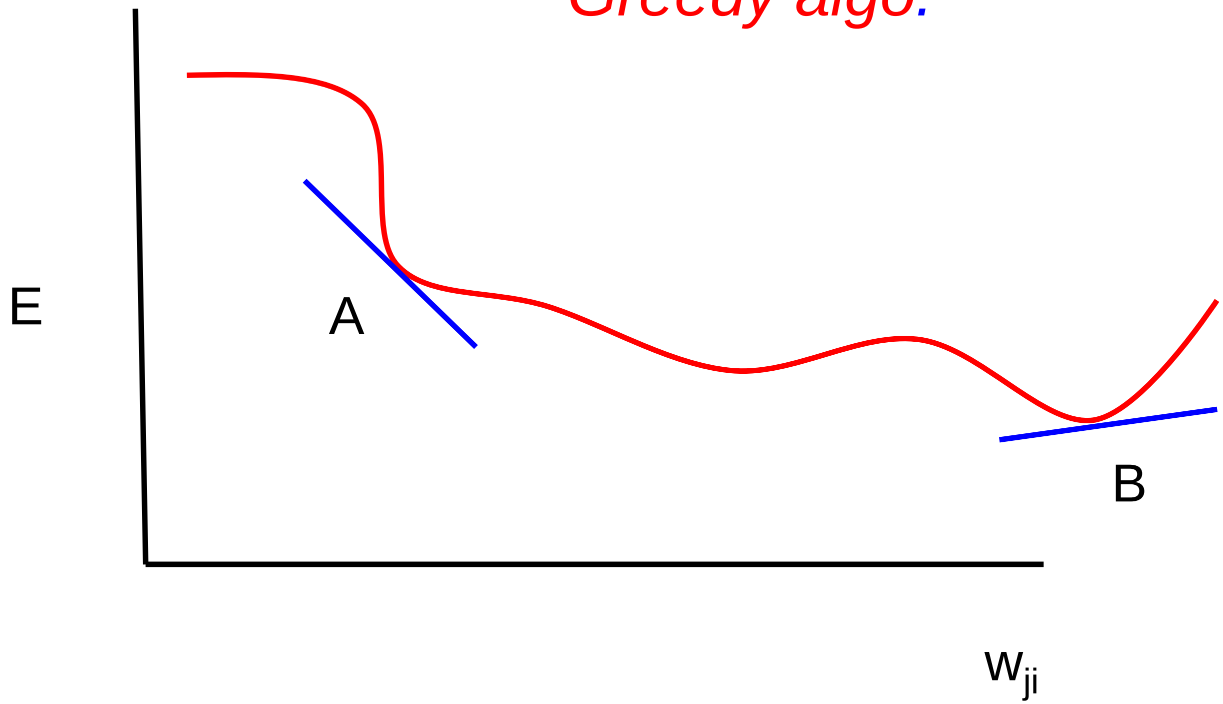
η= *learning rate,*

E=loss, w_{ji}*= weight of connection from the* i^{th} *neuron to* j^{th}

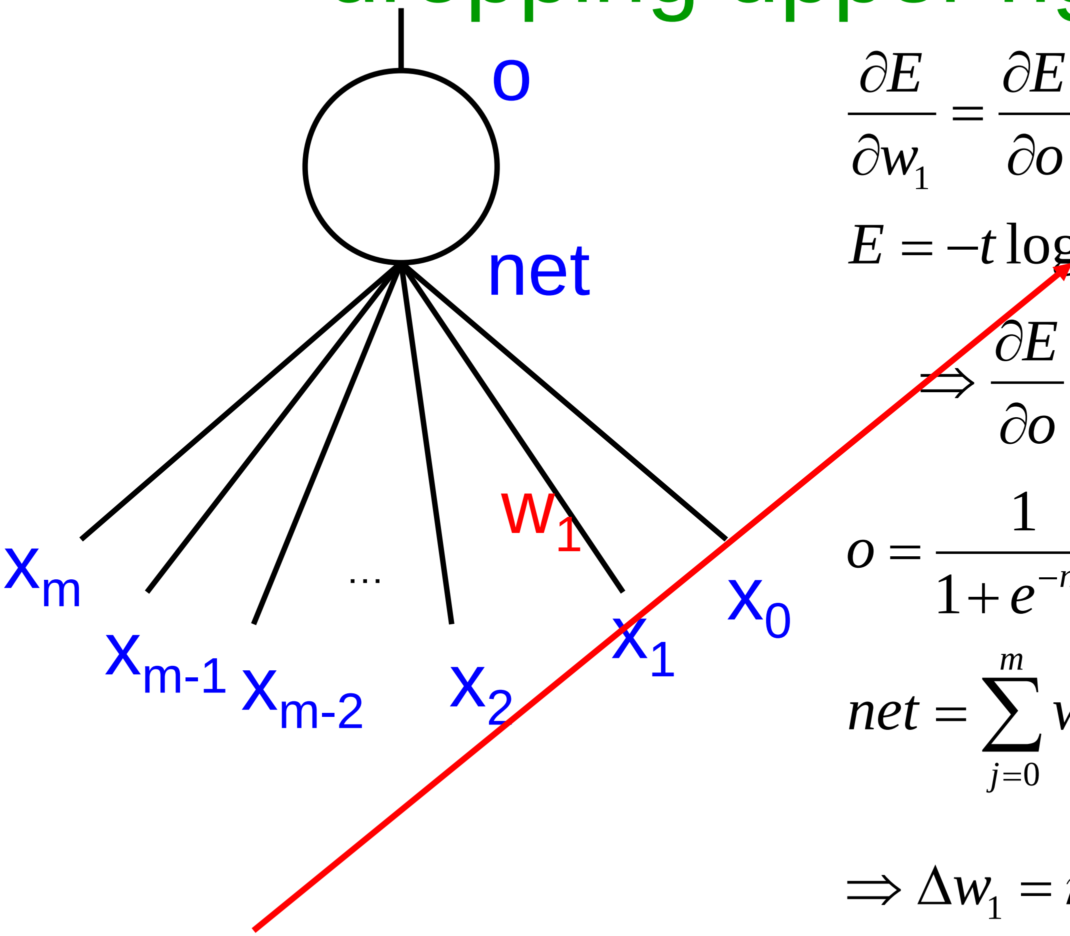
At A, $\delta E / \delta w_{ji}$ *is negative, so* Δw_{ji} *is positive.*

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

E always decreases. Greedy algo.



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



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

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

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

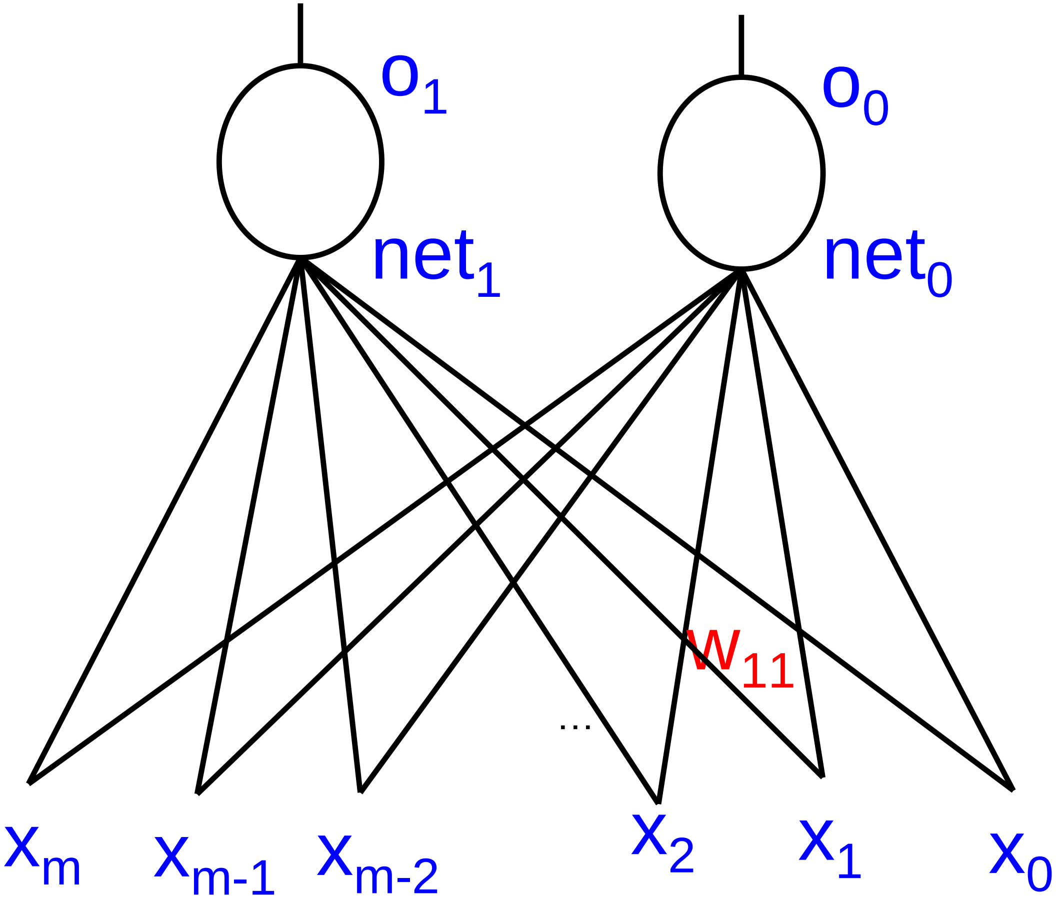
$$o = \frac{1}{1+e^{-net}} \, (sigmoid) \Rightarrow \frac{\partial o}{\partial net} = o(1-o)$$

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

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

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

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)

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

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

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

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

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

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

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

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

Can be generalized

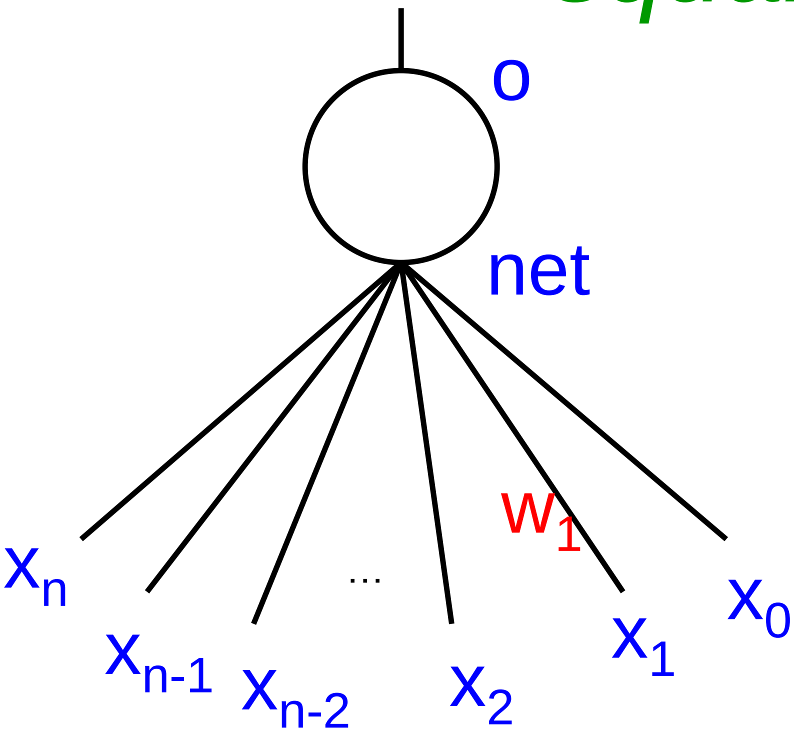
- When E is Cross Entropy Loss

- 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

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

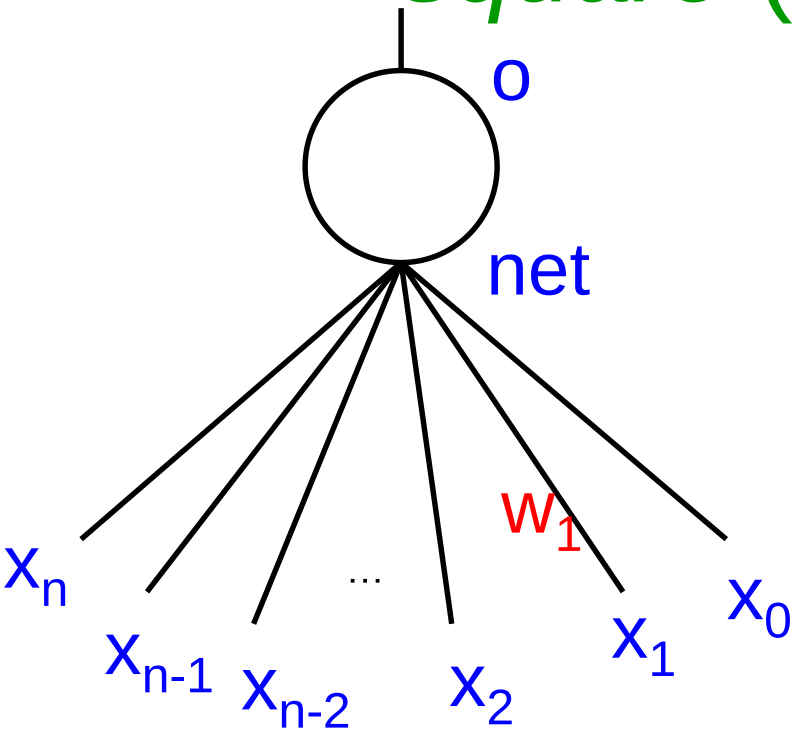
$$L = \frac{1}{2}(t-o)^2 \;\; \Rightarrow \frac{\partial L}{\partial o} = -(t-o) \;\; (1)$$

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

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

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

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



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

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

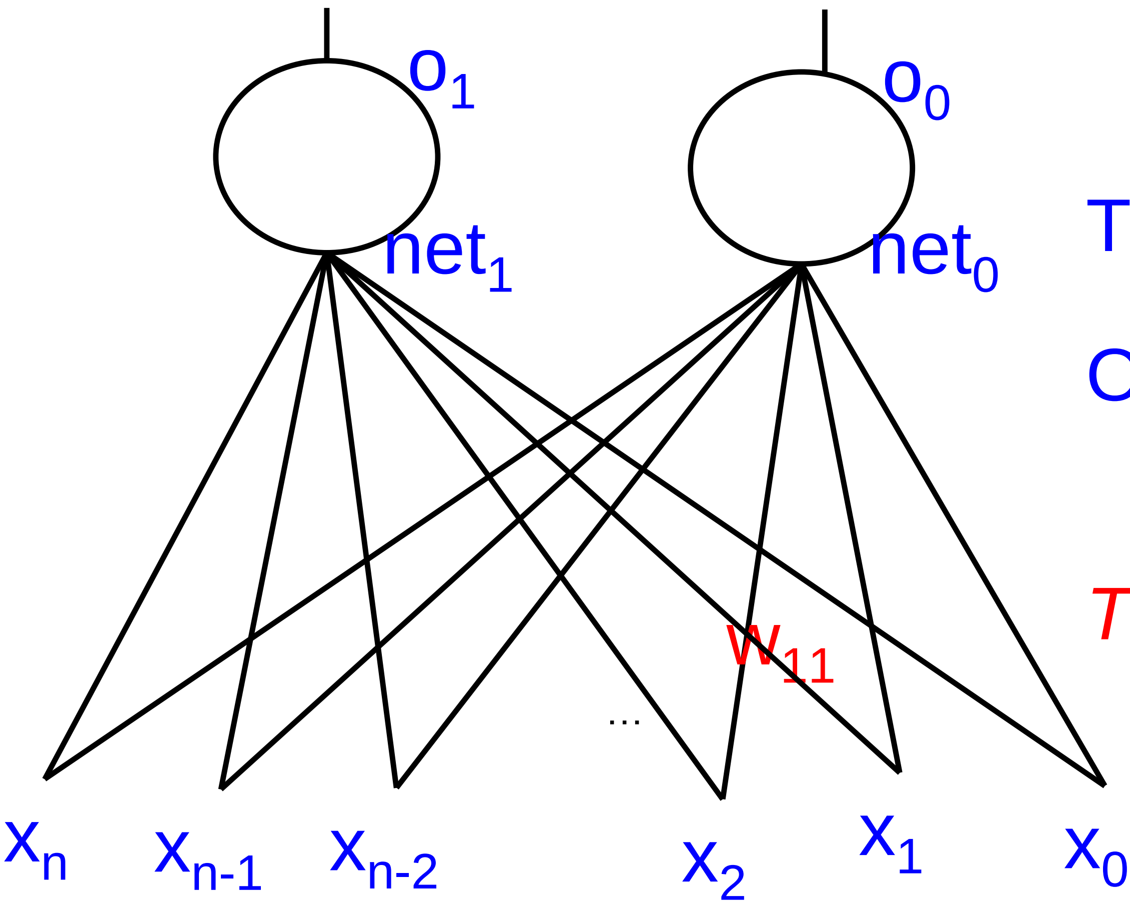
$$o = \frac{1}{1+e^{-net}} \; (sigmoid) \Rightarrow \frac{\partial o}{\partial net} = o(1-o) \; (2)$$

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

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

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

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



Target vector: $<t_1, t_0>$

Observed vector: $<o_1, o_0>$

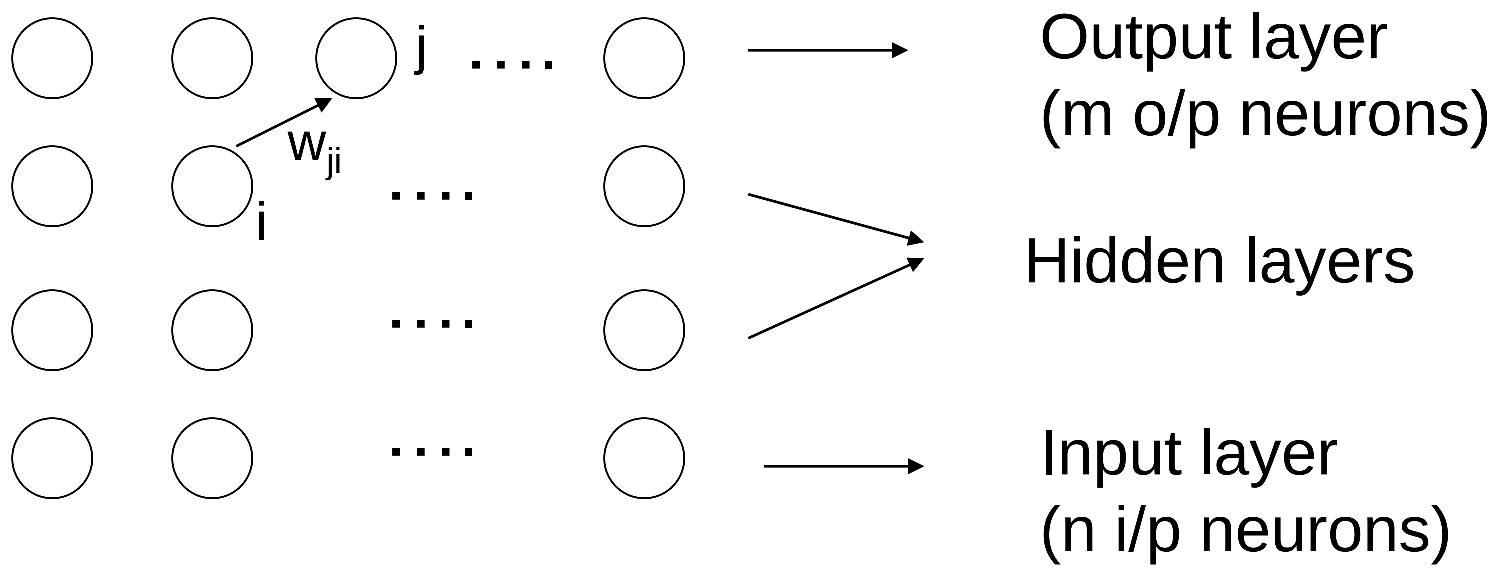
TSS Loss $= \frac{1}{2}[(t_1-o_1)^2+(t_0-o_0)^2]$

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

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

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

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

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

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

Observations from Δw_{ji}

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

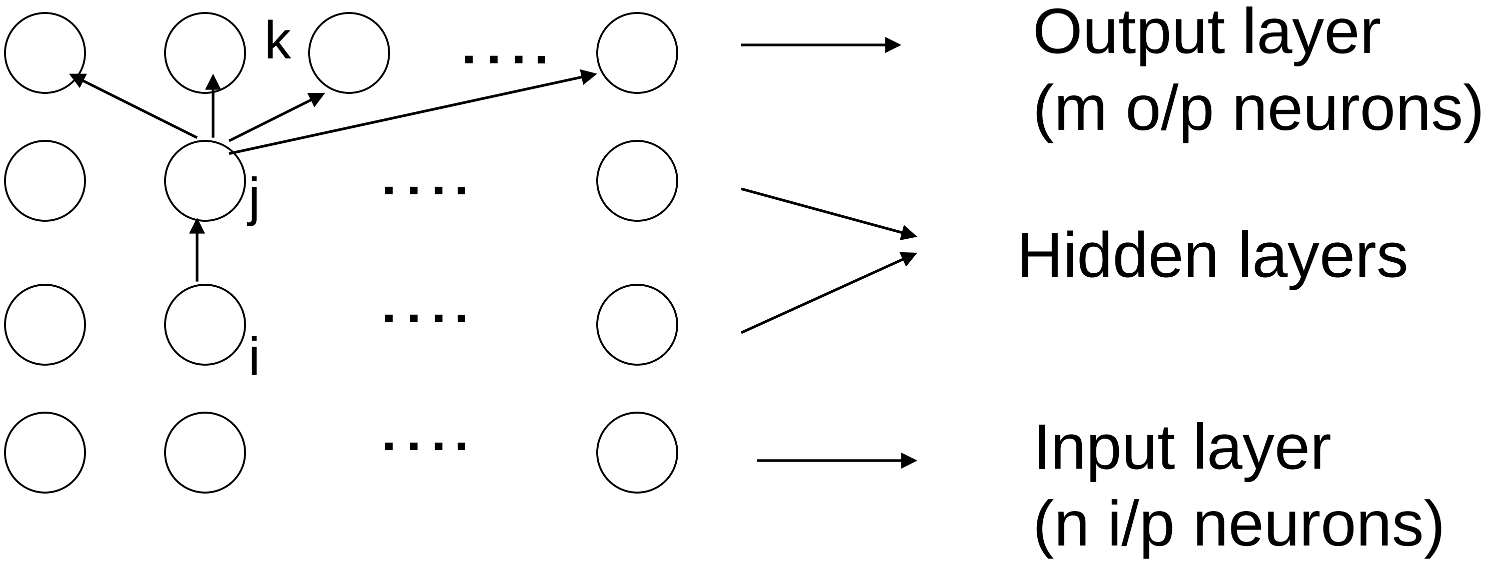
$$\Delta w_{ji} \to 0 \quad \text{if,}$$

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

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

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

Backpropagation for hidden layers



δ_k is propagated backwards to find value of δ_j

Backpropagation – for hidden layers

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

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

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

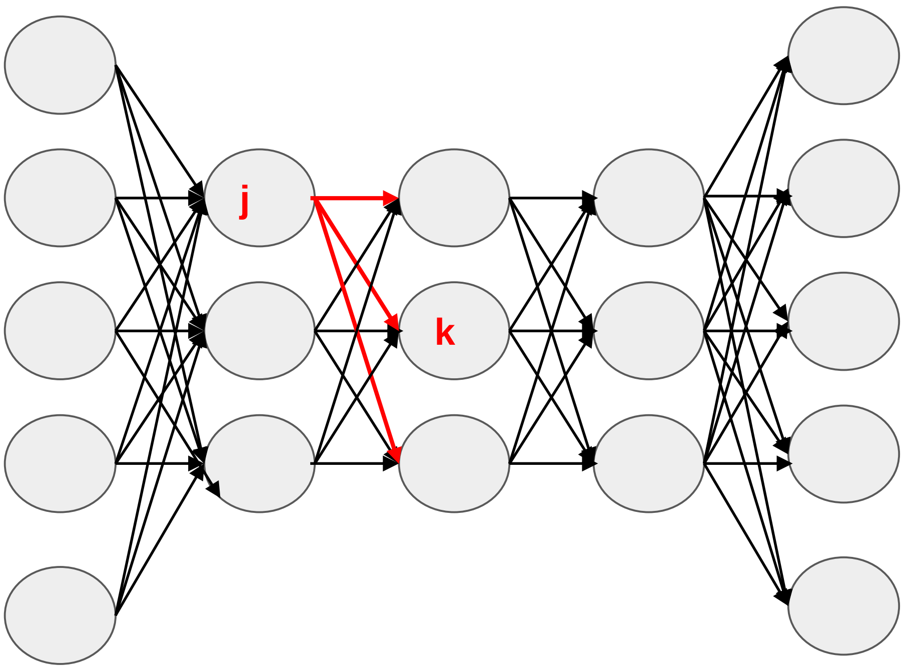
This recursion can give rise to vanishing and exploding Gradient problem

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

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

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

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

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

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