CS772: Deep Learning for Natural Language Processing (DL-NLP)

Perceptron, Sigmoid, Softmax

Pushpak Bhattacharyya
Computer Science and Engineering Department
IIT Bombay

Week 2 of 8th Jan, 2024

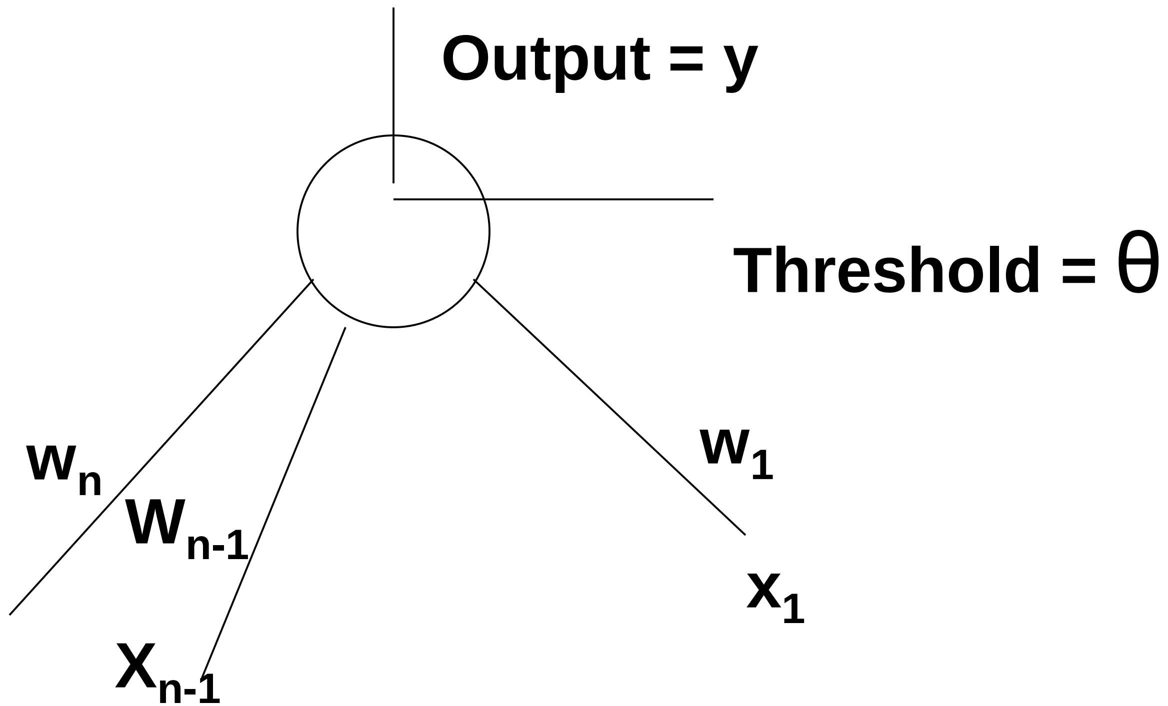
1-slide recap

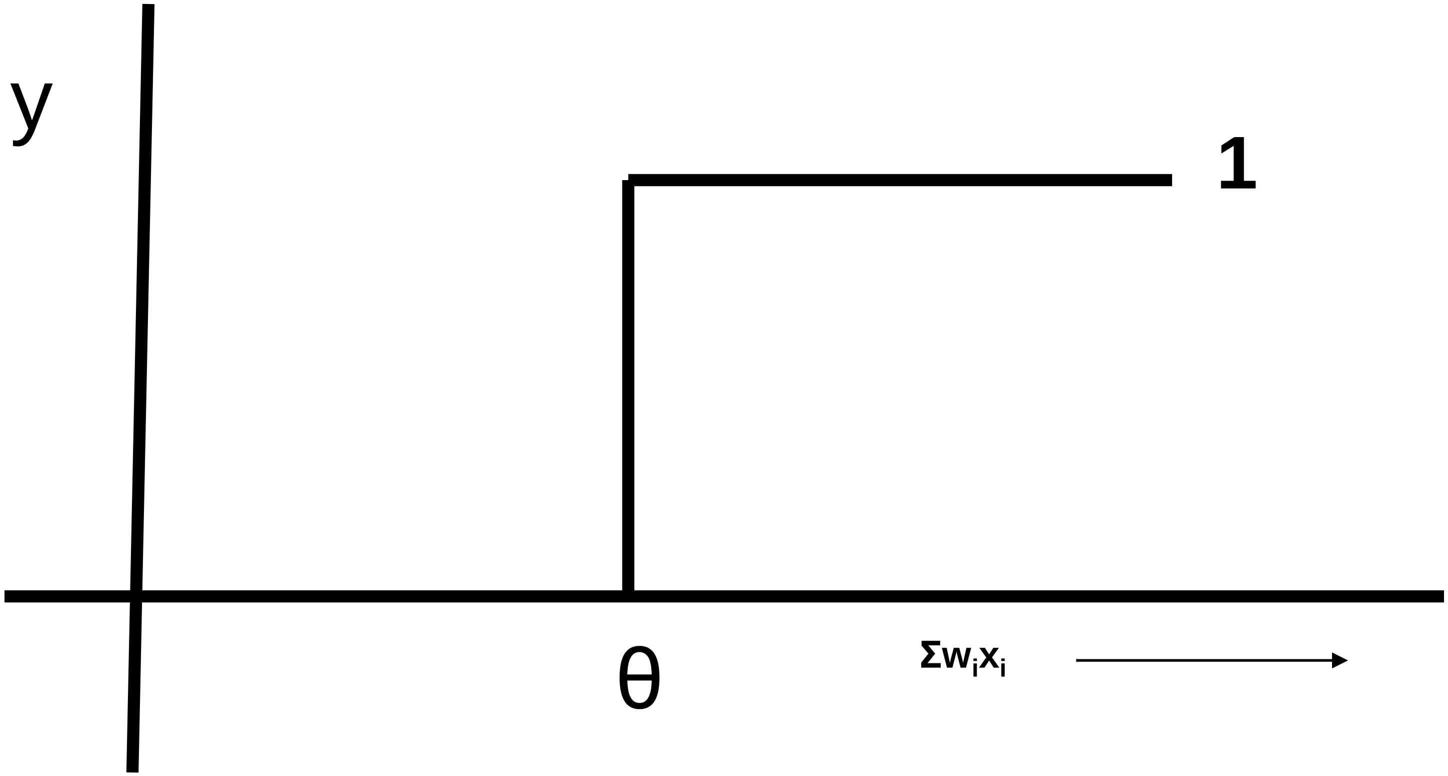
- Nature of language- displacement, recursion etc.
- Neurophysiology- Broca and Wernicke
- Nature of NLP: NLP stack; NLP=linguistics+probability; 3 gens of NLP
- Main Challenge: Ambiguity
- ChatGPT's (an LLM) amazing capability- "Buffalo" sentence
- Course info- evaluation, references
- Heart of ML-NLP: *argmax(P(B|A))*

The Perceptron

The Perceptron Model

- A perceptron is a computing element with input lines having associated weights and the cell having a threshold value. The perceptron model is motivated by the biological neuron.





- Step function / Threshold function

  $y = 1$ for $\Sigma_i w_i x_i >= \theta$

  $= 0$ otherwise

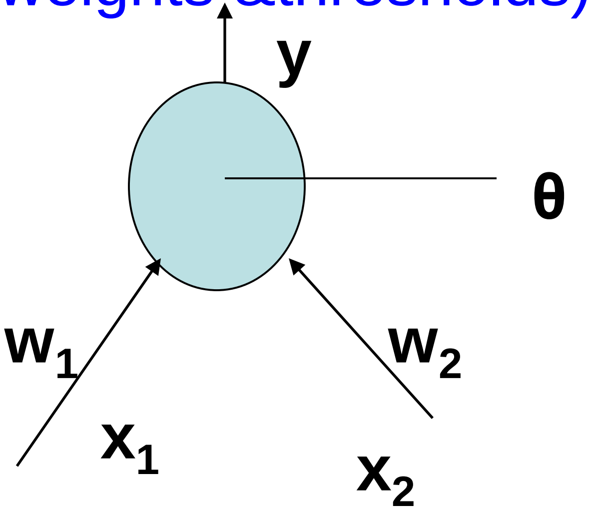
Features of Perceptron

- Input output behavior is discontinuous and the derivative does not exist at $\Sigma_i w_i x_i = \theta$
- $\Sigma_i w_i x_i - \theta$ is the net input denoted as *net*
- Referred to as a linear threshold element - linearity because of x appearing with power 1
- $y = f(net)$: Relation between y and net is non-linear

Computation of Boolean functions: AND

x_2	x_1	**y**
0	0	0
0	1	0
1	0	0
1	1	1

The parameter values (weights &thresholds) need to be found.



Computing parameter values

- $w1 * 0 + w2 * 0 < \theta \Rightarrow \theta > 0$; since $y=0$
- $w1 * 0 + w2 * 1 < \theta \Rightarrow w2 < \theta$; since $y=0$
- $w1 * 1 + w2 * 0 < \theta \Rightarrow w1 < \theta$; since $y=0$
- $w1 * 1 + w2 * 1 >= \theta \Rightarrow w1 + w2 >= \theta$; since $y=1$
- $w1=w2= 0.5$, $\theta=0.9$ is a possibility

Other Boolean functions

OR can be computed using values of

$w1 = w2 = 1$ and $\theta = 0.5$

XOR cannot be computed:

$w1 * 0 + w2 * 0 < \theta \Rightarrow \theta > 0$

$w1 * 0 + w2 * 1 >= \theta \Rightarrow w2 >= \theta$

$w1 * 1 + w2 * 0 >= \theta \Rightarrow w1 >= \theta$

$w1 * 1 + w2 * 1 < \theta \Rightarrow w1 + w2 < \theta$

No set of parameter values satisfy these inequalities.

Threshold functions

- N variables: # Boolean functions (2^{2^n}); #Threshold Functions (2^{n^2})
- 1 4 4
- 2 16 14
- 3 256 128
- 4 64K 1008
- Functions computable by perceptrons- threshold functions, #TF becomes negligibly small for larger values of #BF.
- For n=2, all functions except XOR and XNOR are computable.

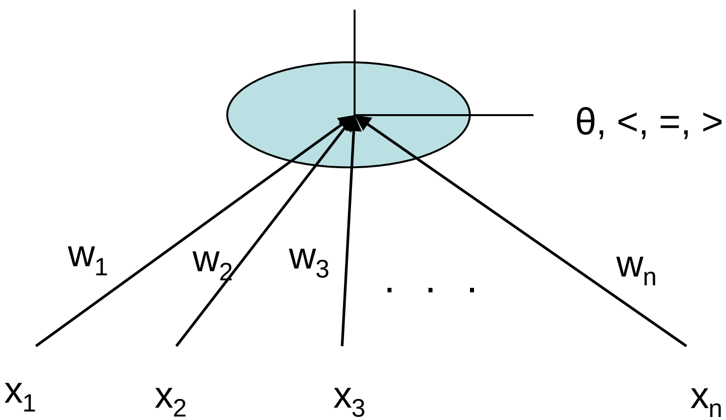
Perceptron Training Algorithm (PTA)

Preprocessing:

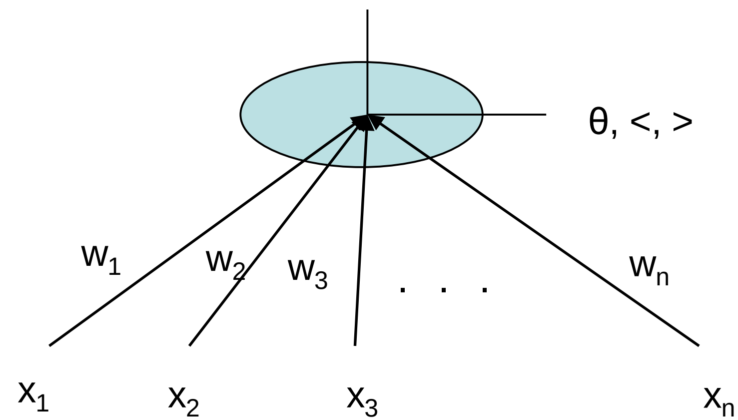
1. The computation law is modified to

$$y=1 \text{ if } \sum w_i x_i > \theta \Rightarrow \sum w_i x_i - \theta > 0$$

$$y=0 \text{ if } \sum w_i x_i < \theta \Rightarrow \sum w_i x_i - \theta < 0$$



→

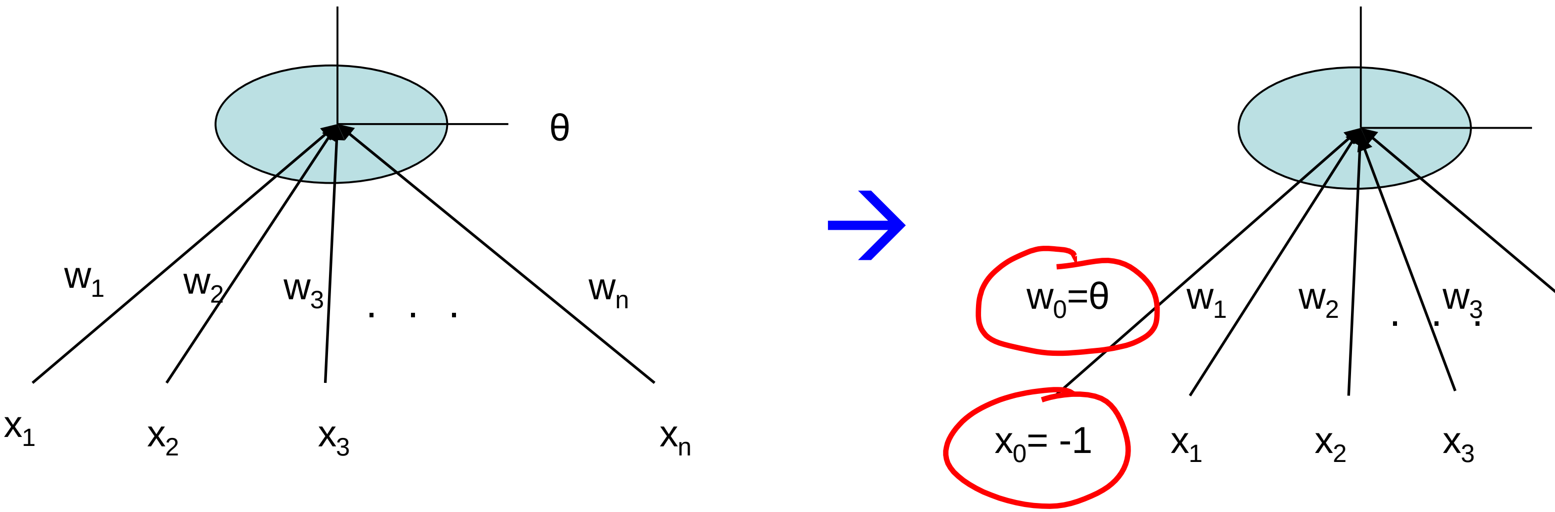


PTA – preprocessing cont…

2. Absorb θ as a weight

$\Rightarrow \sum w_i x_i - \theta < 0$

$\Rightarrow -\sum w_i x_i + \theta > 0$



Hence we negate the 0, class example.

3. Negate all the zero-class examples

Example to demonstrate preprocessing

- **OR perceptron**

1-class <1,1> , <1,0> , <0,1>

0-class <0,0>

Some of the pre-processing steps that are necessary.

Augmented x vectors:-

1-class <-1,1,1> , <-1,1,0> , <-1,0,1>

0-class <-1,0,0>

Negate 0-class:- <1,0,0>

Example to demonstrate preprocessing cont..

Now the vectors are

Augmented vectors.

	x_2	x_1	x_0
X_0	0	0	-1
X_1	0	1	-1
X_2	1	0	-1
X_3	1	1	-1

After negating the 0-class

preprocessed vectors.

	x_2	x_1	x_0
X_0	0	0	1
X_1	0	1	-1
X_2	1	0	-1
X_3	1	1	-1

Perceptron Training Algorithm

1. Start with a random value of w

   ex: <0,0,0…>

2. Test for $WX_i > 0$

   only this condition is sufficient for the seperability of the perceptron.

   If the test succeeds for i=1,2,…n

   then return W

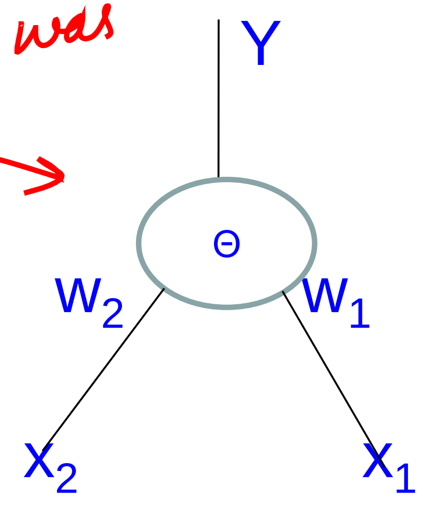
3. Modify W, $W_{next}=W_{prev}+X_{fail}$

PTA on NAND

NAND:

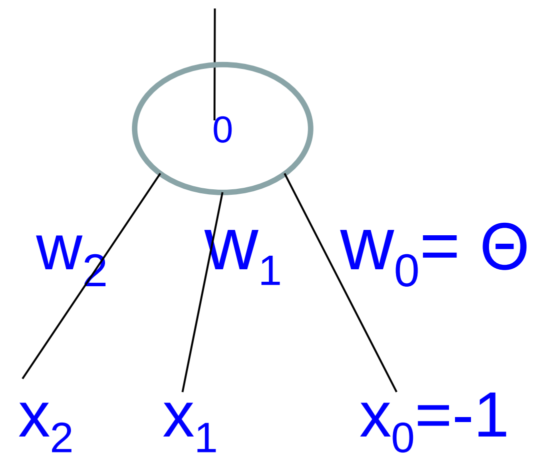
X2	X1	Y
0	0	1
0	1	1
1	0	1
1	1	0

The threshold was θ before



Now, it is converted to 0 after the θ is absorbed.

Converted To



Preprocessing

NAND Augmented:

x_2	x_1	x_0	Y
0	0	-1	1
0	1	-1	1
1	0	-1	1
1	1	-1	0

NAND-0 class Negated

	x_2	x_1	x_0
X_0:	0	0	-1
X_1:	0	1	-1
X_2:	1	0	-1
X_3:	-1	-1	1

Vectors for which $W=<w_2\ w_1\ w_0>$ has to be found such that $W. X_i > 0$

PTA Algo steps

Step-0: $W_0 = <0, 0, 0>$

$W_1 = <0, 0, 0> + <0, 0, -1>$ {X_0 Fails}

$= <0, 0, -1>$

$W_2 = <0, 0, -1> + <-1, -1, 1>$ {X_3 Fails}

$= <-1, -1, 0>$

$W_3 = <-1, -1, 0> + <0, 0, -1>$ {X_0 Fails}

$= <-1, -1, -1>$

$W_4 = <-1, -1, -1> + <0, 1, -1>$ {X_1 Fails}

$= <-1, 0, -2>$

X_0:	0	0	-1
X_1:	0	1	-1
X_2:	1	0	-1
X_3:	-1	-1	1

Trying convergence

$W_5 = <-1, 0, -2> + <-1, -1, 1>$ {X_3 Fails}

$= <-2, -1, -1>$

$W_6 = <-2, -1, -1> + <0, 1, -1>$ {X_1 Fails}

$= <-2, 0, -2>$

$W_7 = <-2, 0, -2> + <1, 0, -1>$ {X_0 Fails}

$= <-1, 0, -3>$

$W_8 = <-1, 0, -3> + <-1, -1, 1>$ {X_3 Fails}

$= <-2, -1, -2>$

$W_9 = <-2, -1, -2> + <1, 0, -1>$ {X_2 Fails}

$= <-1, -1, -3>$

X_0:	0	0	-1
X_1:	0	1	-1
X_2:	1	0	-1
X_3:	-1	-1	1

Trying convergence

W10 = <-1, -1, -3> + <-1, -1, 1> {X3 Fails}
= <-2, -2, -2>

W11 = <-2, -2, -2> + <0, 1, -1> {X1 Fails}
= <-2, -1, -3>

W12 = <-2, -1, -3> + <-1, -1, 1> {X3 Fails}
= <-3, -2, -2>

W13 = <-3, -2, -2> + <0, 1, -1> {X1 Fails}
= <-3, -1, -3>

W14 = <-3, -1, -3> + <0, 1, -1> {X2 Fails}
= <-2, -1, -4>

X_0:	0	0	-1
X_1:	0	1	-1
X_2:	1	0	-1
X_3:	-1	-1	1

$W15 = <-2, -1, -4> + <-1, -1, 1>$ $\{X_3 \text{ Fails}\}$

$= <-3, -2, -3>$

$W16 = <-3, -2, -3> + <1, 0, -1>$ $\{X_2 \text{ Fails}\}$

$= <-2, -2, -4>$

$W17 = <-2, -2, -4> + <-1, -1, 1>$ $\{X_3 \text{ Fails}\}$

$= <-3, -3, -3>$

$W18 = <-3, -3, -3> + <0, 1, -1>$ $\{X_1 \text{ Fails}\}$

$= <-3, -2, -4>$

$W2 = -3$, $W1 = -2$, $W0 = \Theta = -4$

X_0:	0	0	-1
X_1:	0	1	-1
X_2:	1	0	-1
X_3:	-1	-1	1

Succeeds for all vectors

PTA convergence

Statement of Convergence of PTA

- Statement:

Whatever be the initial choice of weights and whatever be the vector chosen for testing, PTA converges if the vectors are from a linearly separable function.

Important

Proof of Convergence of PTA

- Suppose w_n is the weight vector at the n^{th} step of the algorithm.
- At the beginning, the weight vector is w_0
- Go from w_i to w_{i+1} when a vector X_j fails the test $w_i X_j > 0$ and update w_i as

$$w_{i+1} = w_i + X_j$$

- Since Xjs form a linearly separable function,
- there exits w* s.t. $w^* X_j > 0$ for all j

Proof of Convergence of PTA (cntd.)

- Consider the expression

$$G(w_n) = \frac{w_n \cdot w^*}{|w_n|}$$

where w_n = weight at nth iteration

- $G(w_n) = \dfrac{|w_n| \cdot |w^*| \cdot \cos\theta}{|w_n|}$

where □ = angle between w_n and w^*

- $G(w_n) = |w^*| \cdot \cos\theta$ *
- $G(w_n) \leq |w^*|$ (as $-1 \leq \cos\theta \leq 1$)

* $|w^*|$

$\Rightarrow -|w^*| < 0 < |w^*|$

$-1 < 0 < 1$

since, cosθ's value changes accordingly.

Hence,

$G(w_n) \leq |w^*|$

Behavior of Numerator of G

$w_n \, . \, w^* \;=\; (w_{n-1} + X^{n-1}_{fail}) \, . \, w^*$

$= \; w_{n-1} \, . \, w^* + X^{n-1}_{fail} \, . \, w^*$

$= \; (w_{n-2} + X^{n-2}_{fail}) \, . \, w^* + X^{n-1}_{fail} \, . \, w^* \; .....$

$= \; w_0 \, . \, w^* + (X^0_{fail} + X^1_{fail} + ... + X^{n-1}_{fail}). \, w^*$

$w^*.X^i_{fail}$ is always positive: note carefully

$w^* \rightarrow$ optimal weight and hence $w^* x^i_{fail} > 0$

- Suppose $w^*.X^i_{fail} \geq \delta_{min}$, where δ_{min} is a positive quantity
- Num of G $\geq |w_0 \, . \, w^*| + n \, \delta_{min}$

since there were n such failures.

- So, numerator of G grows with n.

Behavior of Denominator of G

- $|w_n| = (w_n \cdot w_n)^{1/2}$

$= [(w_{n-1} + X^{n-1}_{fail})^2]^{1/2}$

$= [(w_{n-1})^2 + 2.\, w_{n-1}.\, X^{n-1}_{fail} + (X^{n-1}_{fail})^2]^{1/2}$

$\leq [(w_{n-1})^2 + (X^{n-1}_{fail})^2]^{1/2}$ (as $w_{n-1}.\, X^{n-1}_{fail} \leq 0$)

$\leq [(w_0)^2 + (X^0_{fail})^2 + (X^1_{fail})^2 + \ldots. + (X^{n-1}_{fail})^2]^{1/2}$

- $|X_j| \leq \delta_{max}$ (max magnitude)
- So, Denom $\leq [(w_0)^2 + n\, \delta_{max}{}^2)]^{1/2}$
- Denom grows as $n^{1/2}$

Some Observations

- Numerator of G grows as n
- Denominator of G grows as $n^{1/2}$

  => Numerator grows faster than denominator
- If PTA does not terminate, $G(w_n)$ values will become unbounded.

Some Observations contd.

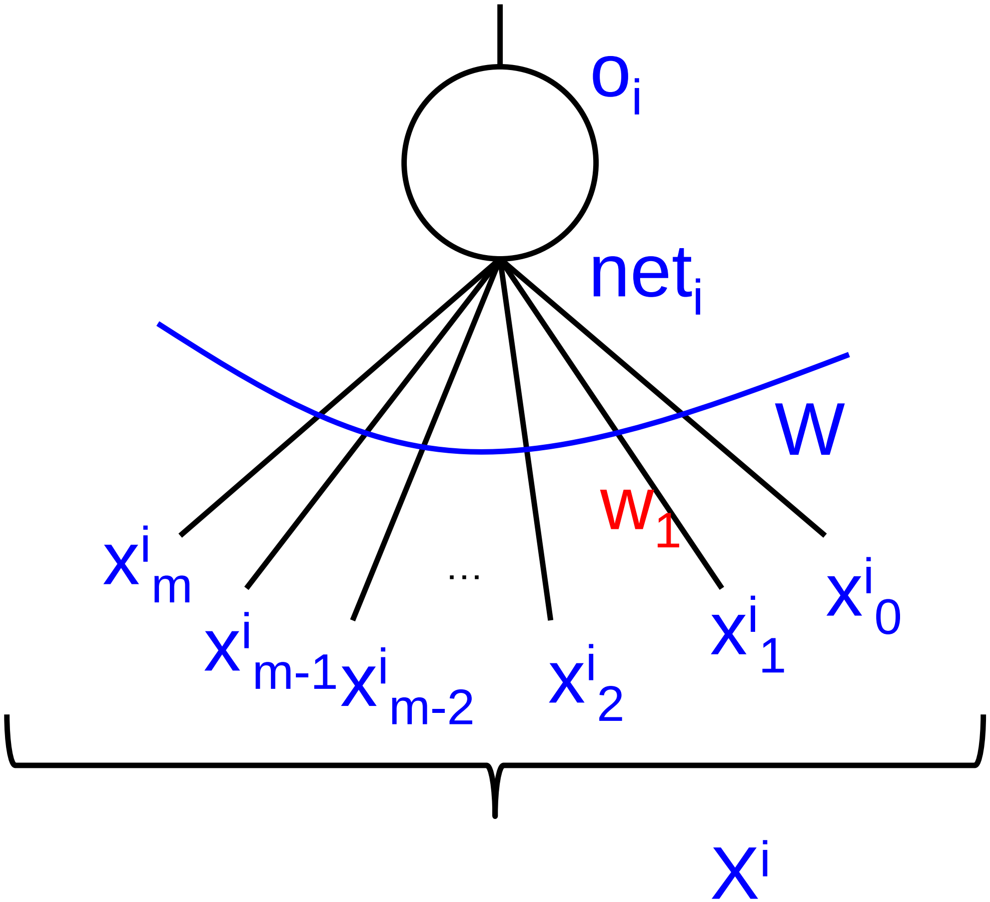
- But, as $|G(w_n)| \leq |w^*|$ which is finite, this is impossible!
- Hence, PTA has to converge.
- Proof is due to Marvin Minsky.

Convergence of PTA proved

- *Whatever be the initial choice of weights and whatever be the vector chosen for testing, PTA converges if the vectors are from a linearly separable function.*

Sigmoid

Sigmoid neuron



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

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

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

$\frac{d}{dx}(x - x^2) = 0$

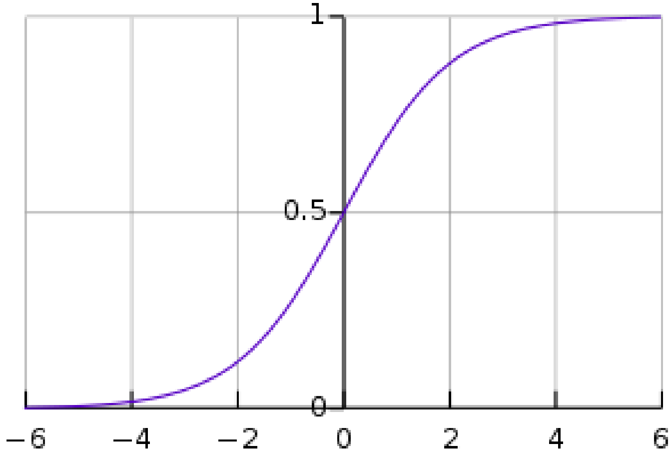
$1 - 2x = 0$

$\therefore \boxed{x = \frac{1}{2}}$

Note → The derivative of the sigmoid function is maximum at 0.5. The maximum value of the derivative is ⇒ $\sigma(x)(1-\sigma(x)) \Rightarrow \frac{1}{2} \times (1 - \frac{1}{2})$

$\Rightarrow \frac{1}{2} \times \frac{1}{2}$

$\Rightarrow 0.25$



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

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

$$\frac{df(x)}{dx} = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$$

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

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

$$= f(x).(1 - f(x))$$

Decision making under sigmoid

- Output of sigmod is between 0-1
- Look upon this value as probability of Class-1 (C_1)
- $1\text{-}sigmoid(x)$ 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~~ 0 is for class 2 and ~~0.5~~ 0 to plus infinity is class 2.

We cannot do this since we need probability
(across all the classes) distribution which is only possible in case of the
softmax activation function. Also, sigmoid
saturates in $+\infty$ and $-\infty$, so it will be
difficult to find class probability scores accurately.

- Think about it!!

If there are n-classes then we need (n-1) neurons (Independent). for → classification.

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

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

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

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

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

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

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

This is a probability distribution, since all the values adds upto 1, and the values are ≥ 0.

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

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

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.

Theoritical foundation

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

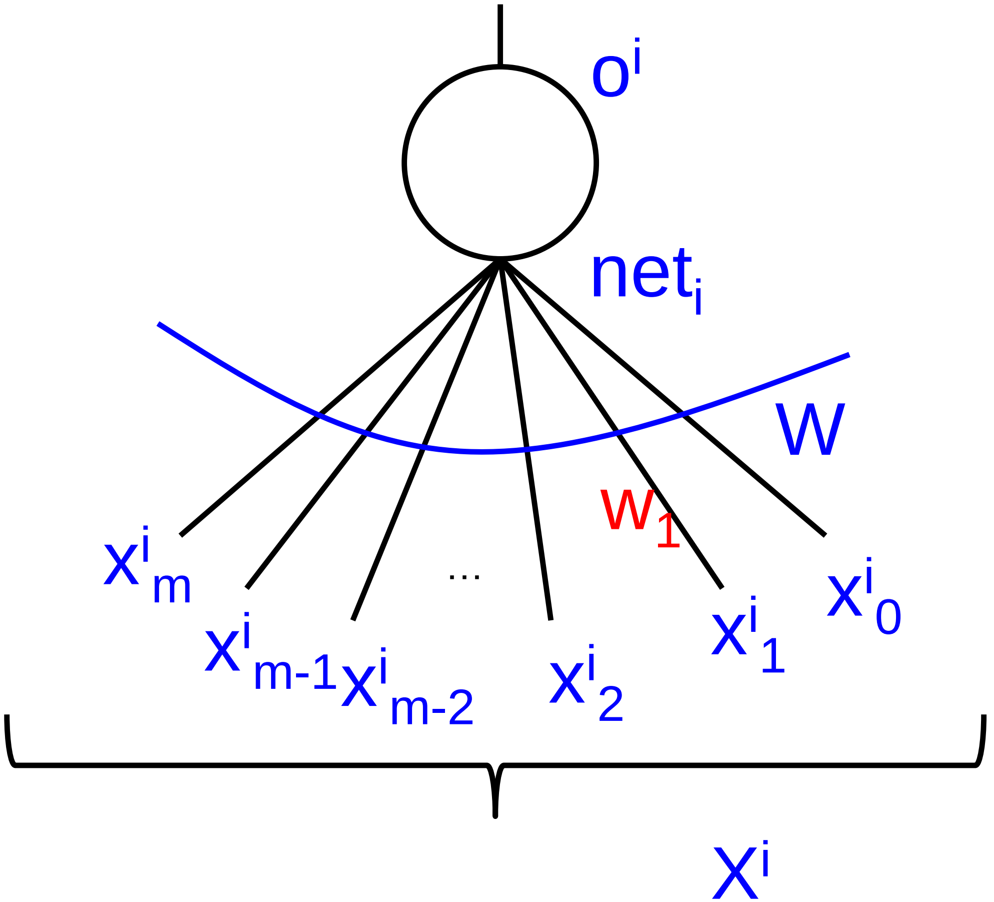
Part of the reality / or the Training data

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

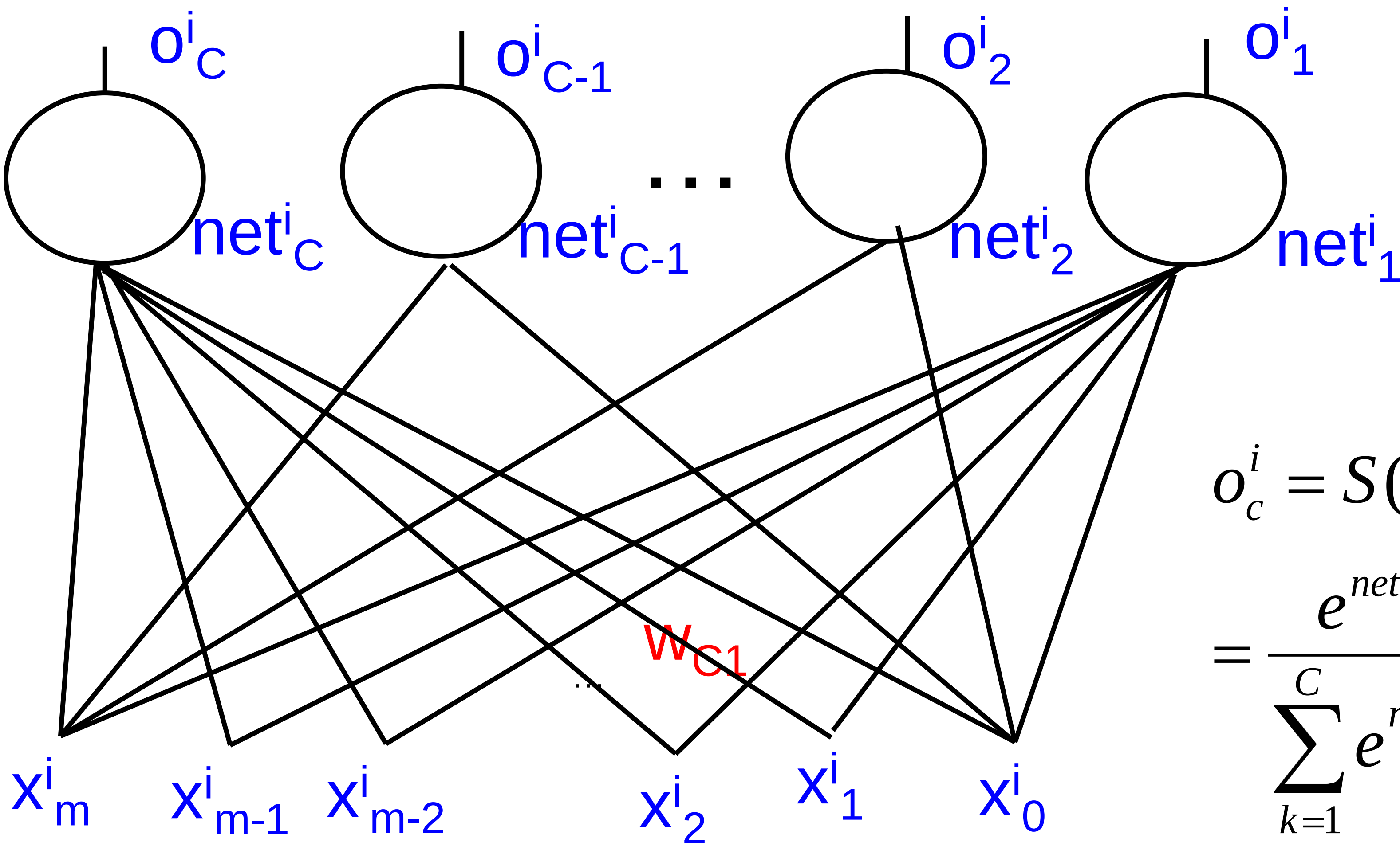
Sigmoid neuron



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

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

Softmax Neuron



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

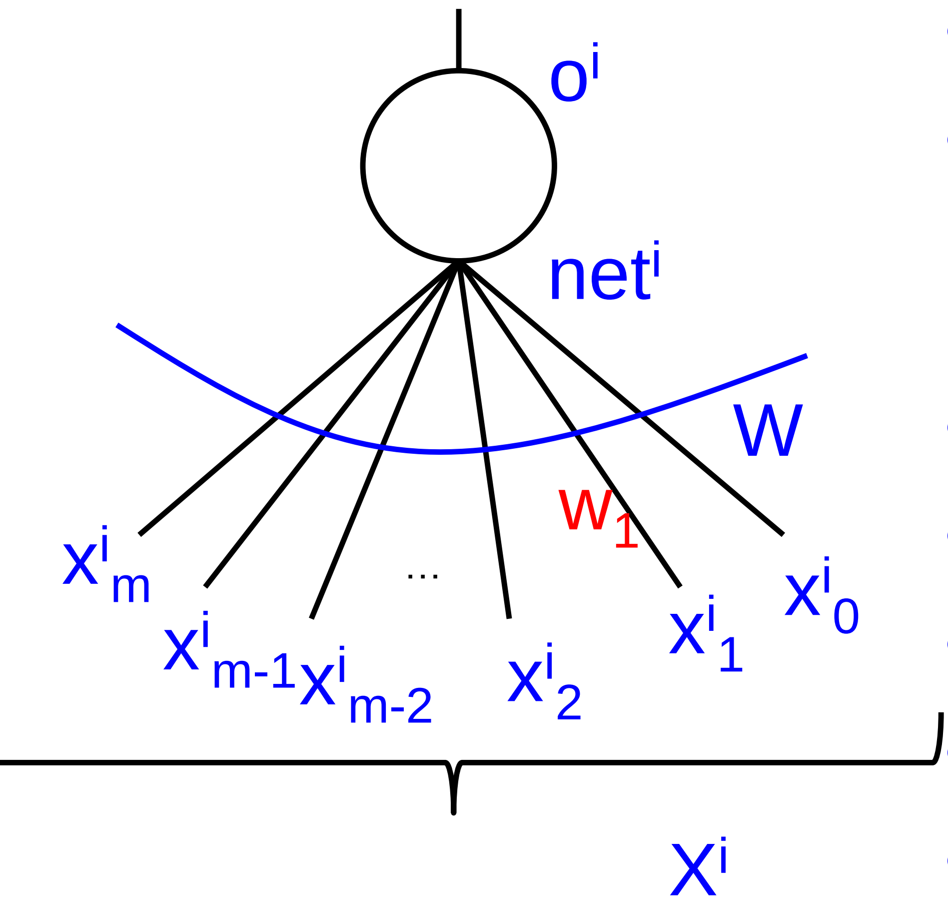
$$= \frac{e^{net_c^i}}{\sum_{k=1}^{C} e^{net_k^i}}$$

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

Notation

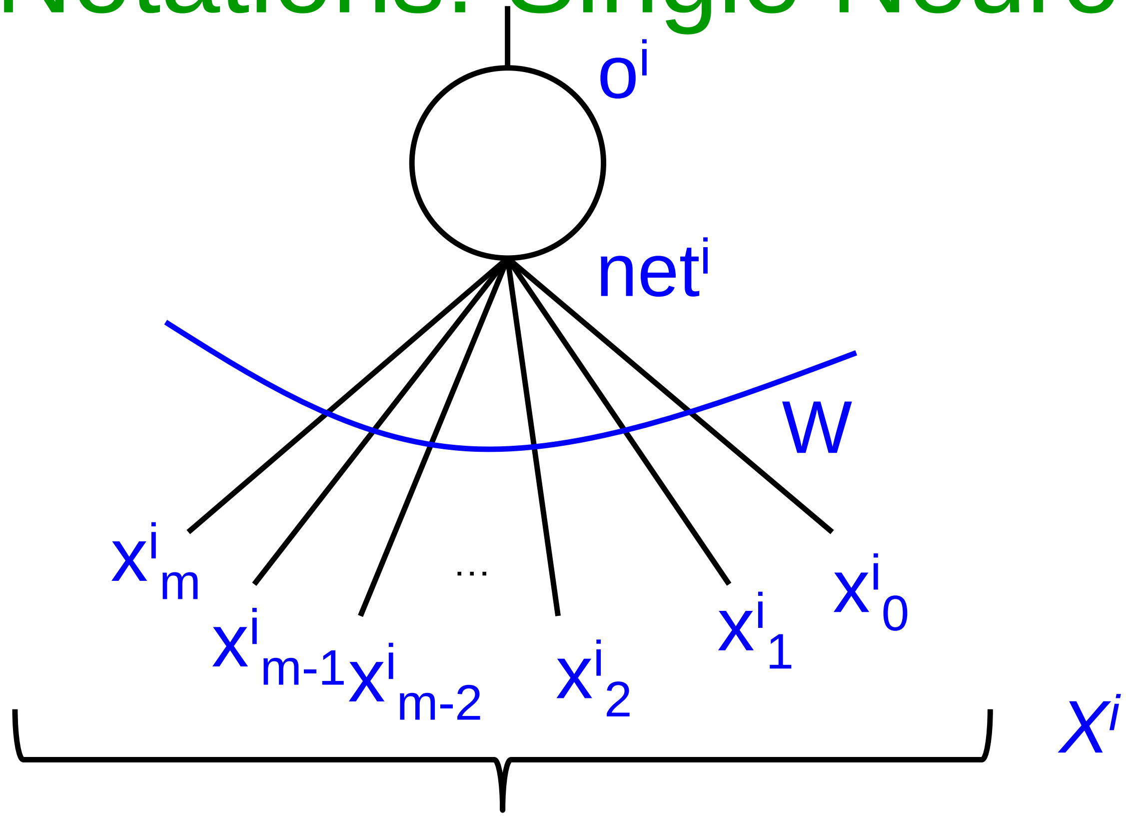
- $i=1..N$
- N i-o pairs, i runs over the training data
- $j=0\ldots m$, m components in the input vector, j runs over the input dimension (also weight vector dimension)
- $k=1\ldots C$, C classes (C components in the output vector)

Fix Notations: Single Neuron (1/2)



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



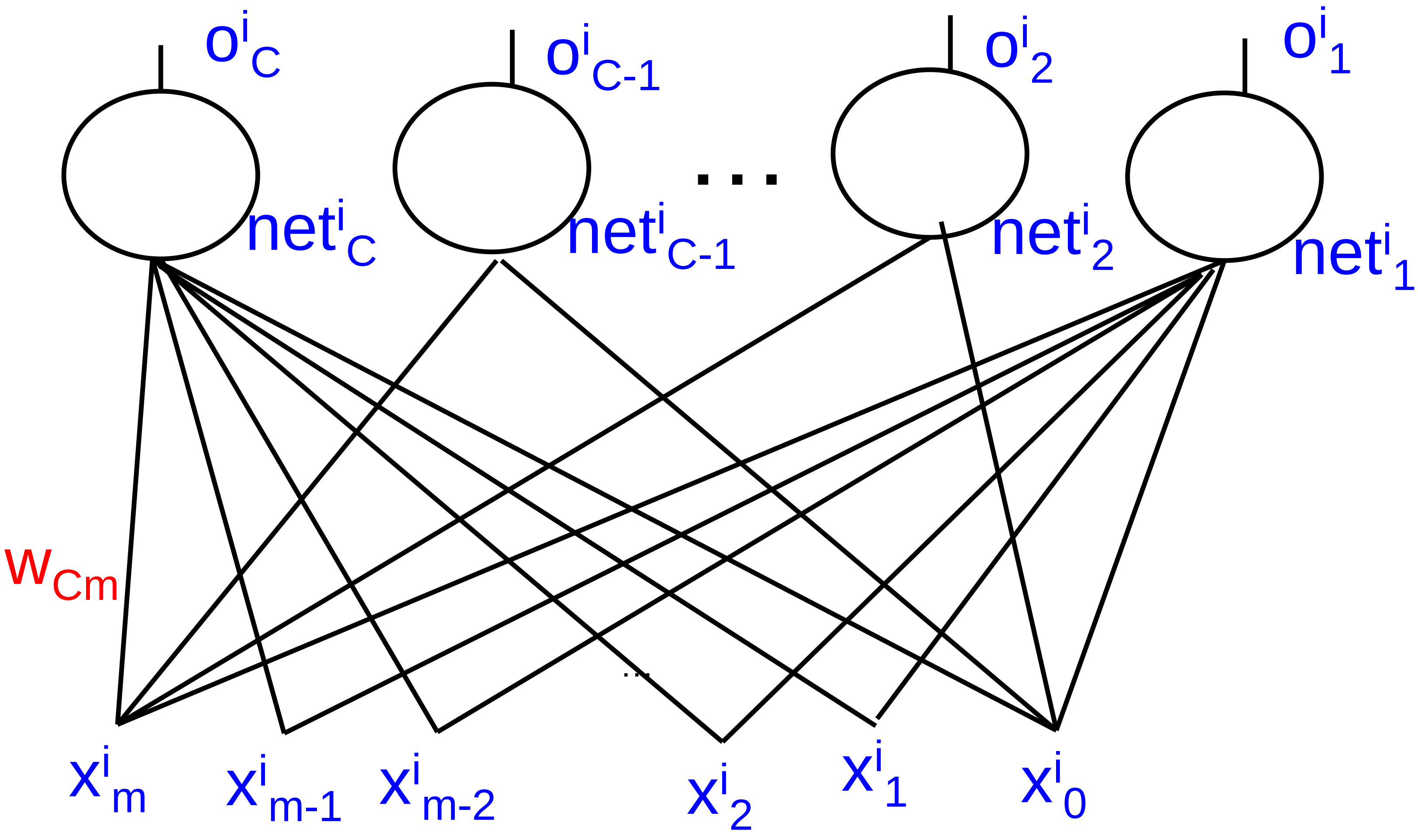
W and each X^i has m components

W:<w_m, w_{m-1}, …, w_2, w_0>

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

Upper suffix i indicates i^{th} input

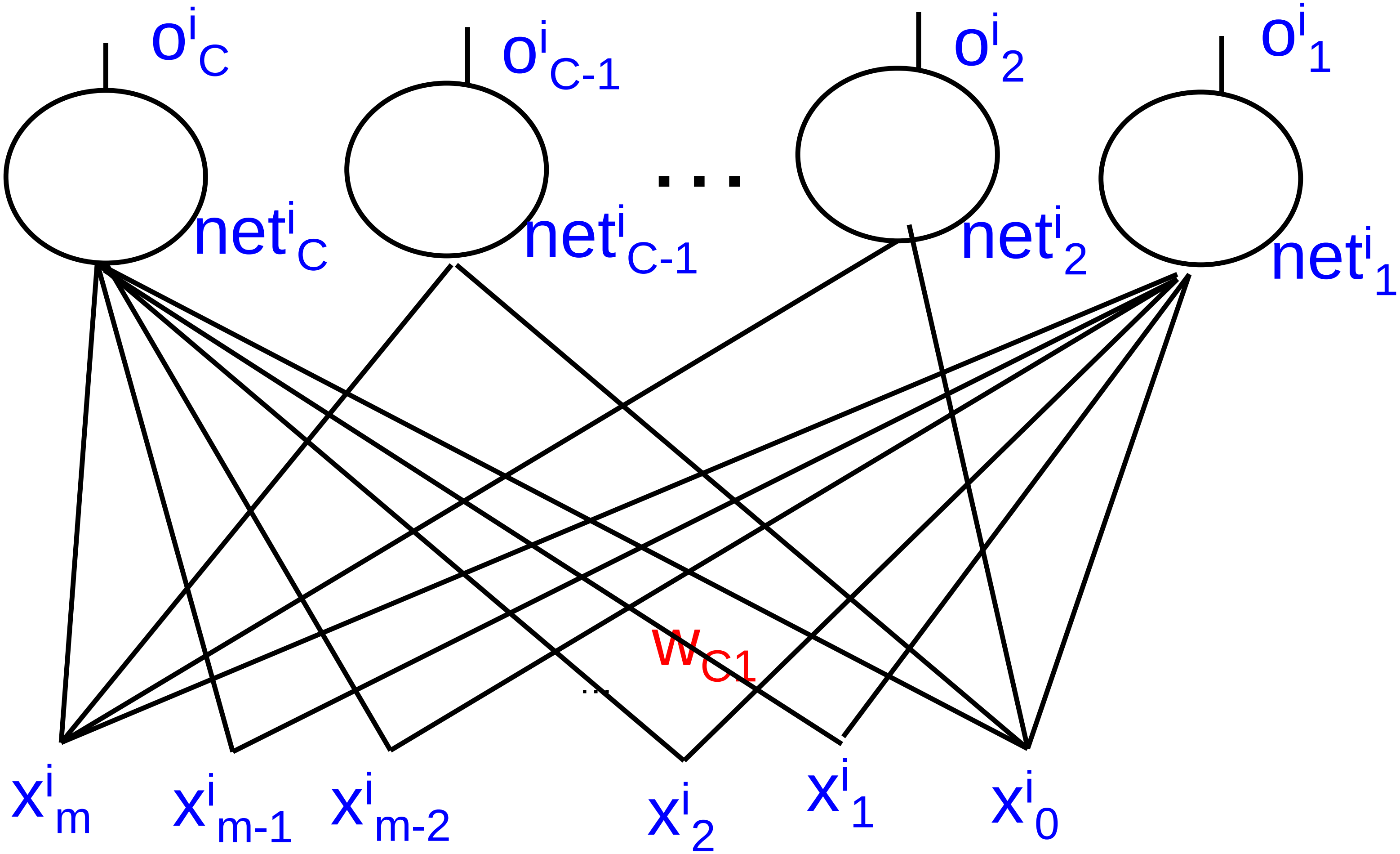
Fixing Notations: Multiple neurons in o/p layer



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

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

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

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

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

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

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

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

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

Here o^i_c & net^i_c's c's are same

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

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

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

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

Always use $net^i_{c'}$ for the derivative / differentiation when they are different.