Cmput 466 / 551

Artificial Neural Networks #1: Backprop

Covering chapter (HTF) 11

+

"An Intro to Conjugate Gradient Method without Agonizing Pain"

R Greiner
Department of Computing Science
University of Alberta

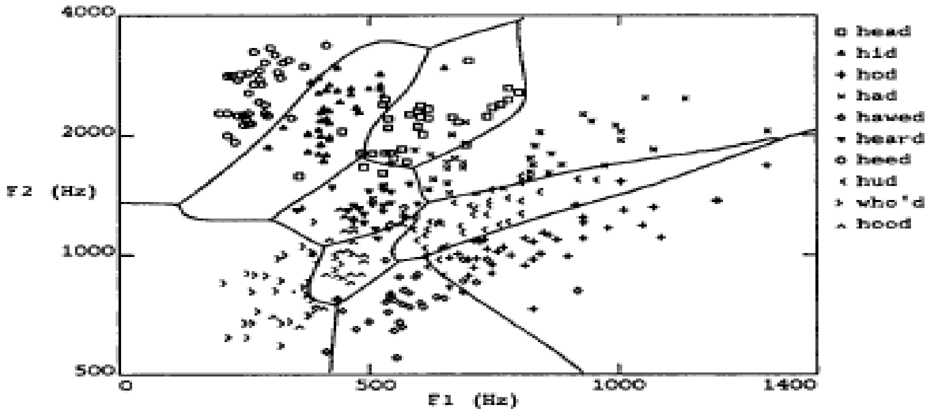
Thanks: T Dietterich, R Parr, J Shewchuk

Outline

- Introduction
  - Historical Motivation, non-LTU, Objective
  - Types of Structures
- Multi-layer Feed-Forward Networks
  - Sigmoid Unit
  - Backpropagation
- Tricks
  - Line Search
  - Conjugate Gradient
  - Alternative Error Functions
- Example: Face Recognition
- Hidden layer representations
- Towards Deeper Nets
- ~~Recurrent Networks~~

Motivation for non-Linear Classifiers

- Linear methods are "weak"
  - Make strong assumptions
  - Can only express relatively simple functions of inputs
  - But …



- Need to learn more-expressive classifiers, that can do more!
  - What does the space of hypotheses look like?
  - How do we navigate in this space?

Non-Linear $\Rightarrow^{?}$ Neural Nets

- Linear separability depends on FEATURES!!

  A function can be
  - not-linearly-separable with one set of features,
  - but linearly separable in another

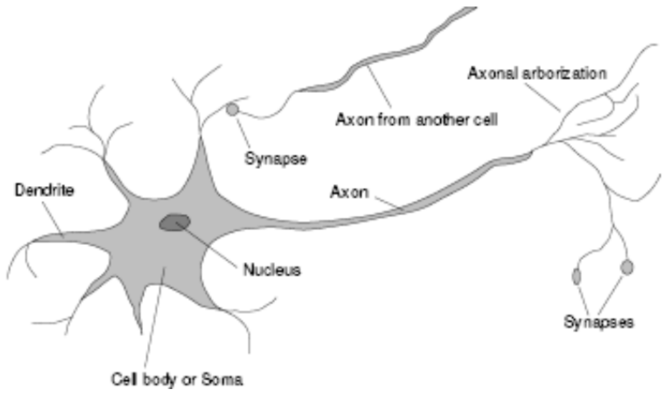
- Want features that make function linearly-separable

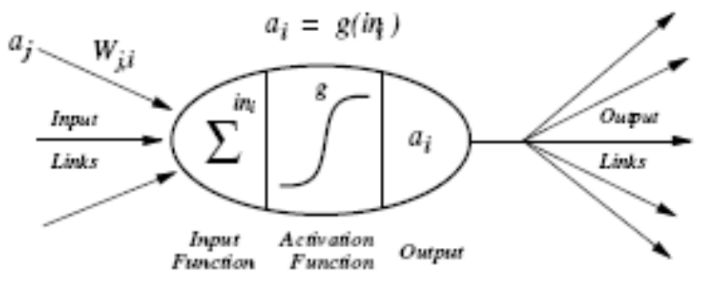
- Approaches:
  - Hand-constructed features?
  - Specified kernels
  - Learned features… (artificial) neural nets ??

Why “Neural Network” ?

- Brains – network of neurons –
  are only known example of actual intelligence
- Built from many individual neurons
  - Individual neurons are slow, boring
  - Brains succeed by using massive parallelism
  - … and sequences – output of one, is input to another
  - Learning: setting the (strengths of) connections
- Idea: **Use many simple “units”**
  for building approximators!
- Raises many issues:
  - Is the computational metaphor suited to the computational hardware?
  - How to copy (only) the important parts?

Natural, vs Artificial, Neurons





Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

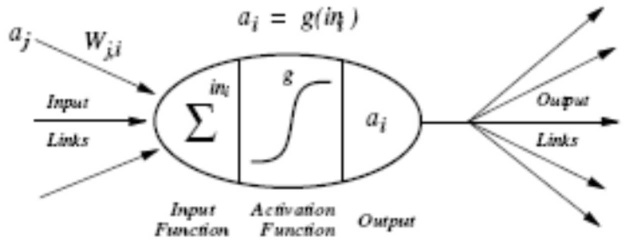
Artificial Neural Networks

- Mathematical abstraction!
- **Units**, connected by **links**; with **weight** $\in \Re$
- Each unit has …

  + set of inputs links from other units

  + set of output links to other units

  … computes activation at next time step



- Lots of simple computational unit

  $\Rightarrow$ massively parallel implementation
- Non-Linear function approximation
  - One of the most widely-used learning methods

"… neural nets are the **second** best thing for learning anything!" J Denker

Artificial Neural Networks

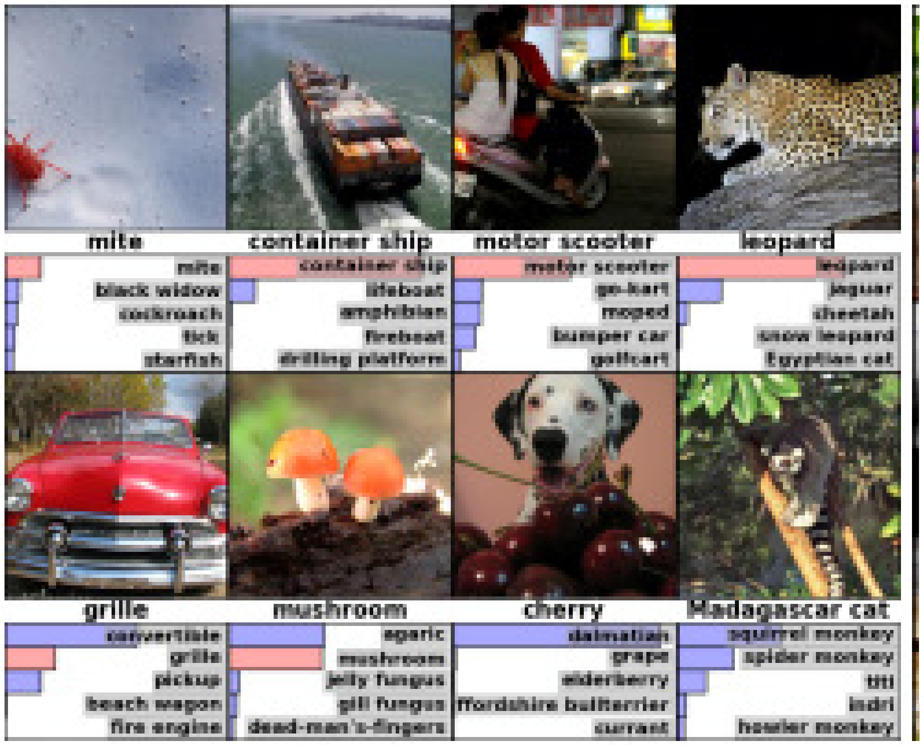
- Rich history, starting in early 40's (McCulloch/Pitts 1943)
- Interests from…

  Neuro-science, Cognitive science, Physics, Statistics, Engineering, CS / EE, ... and AI
- Two views:
  - Modeling the brain
  - "Just" rep'n of complex functions
- Much progress on both fronts

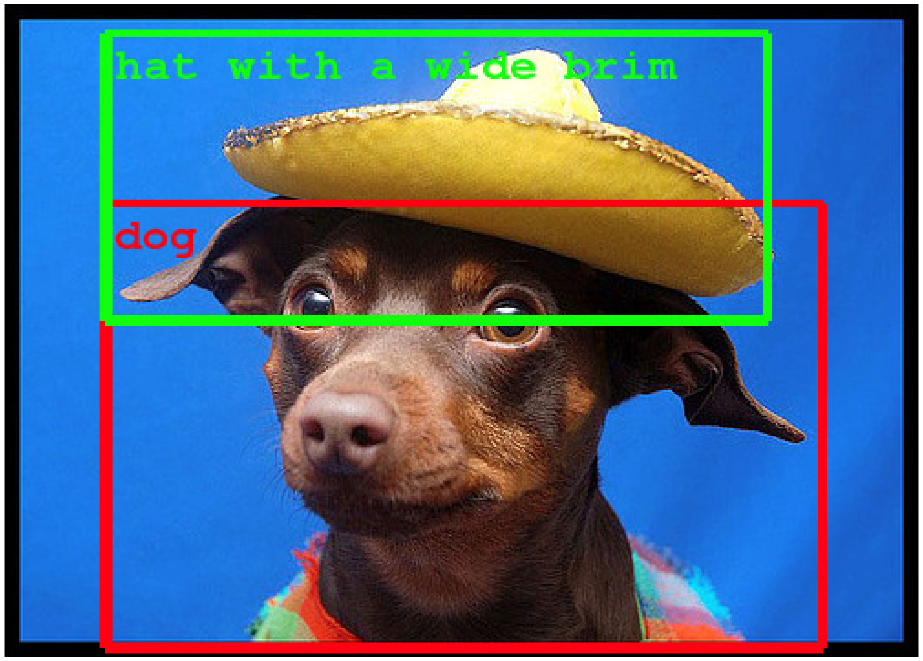
NOW (2016)

Uses of Artificial Neural Nets

- Object Classification in Photographs



AlexNet

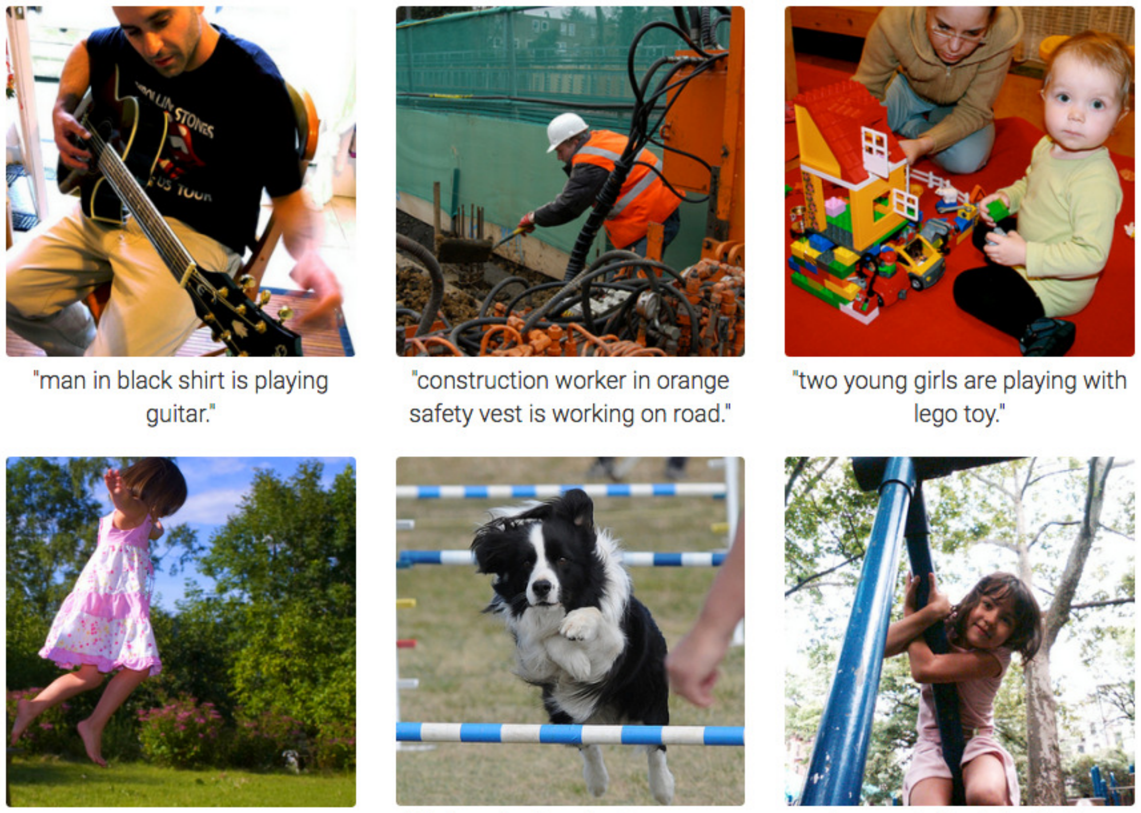


http://machinelearningmastery.com/inspirational-applications-deep-learning/

NOW (2016)

Uses of Artificial Neural Nets

- Object Classification in Photographs
- Image Caption Generation


"man in black shirt is playing guitar."
"construction worker in orange safety vest is working on road."
"two young girls are playing with lego toy."
"girl in pink dress is jumping in air."
"black and white dog jumps over bar."
"young girl in pink shirt is swinging on swing."

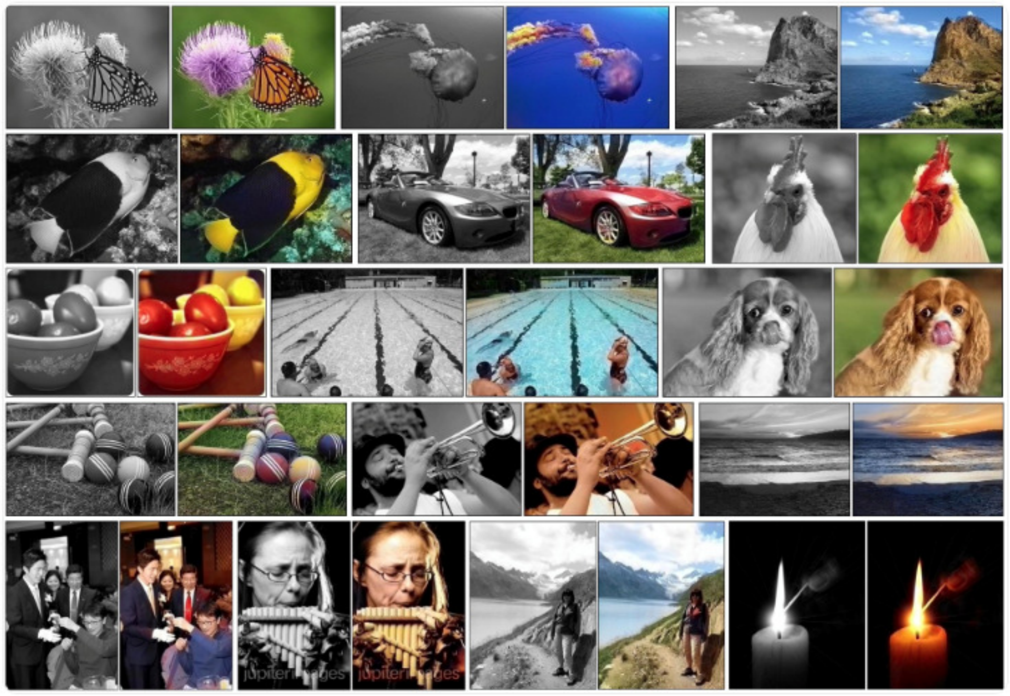
Sample taken from Andrej Karpathy, Li Fei-Fei

http://machinelearningmastery

NOW (2016)

Uses of Artificial Neural Nets

- Object Classification in Photographs
- Image Caption Generation
- Colorization of Black and White Images

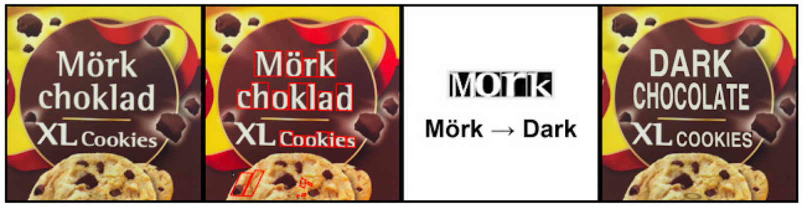


http://machinelearningmastery.com/inspirational-applications-deep-learning/

NOW (2016)

Uses of Artificial Neural Nets

- Object Classification in Photographs
- Image Caption Generation
- Colorization of Black and White Images
- Automatic Machine Translation



http://machinelearningmastery.com/inspirational-applications-deep-learning/

NOW (2016)

Uses of Artificial Neural Nets

- Object Classification in Photographs
- Image Caption Generation
- Colorization of Black and White Images
- Automatic Machine Translation
- Adding Sounds To Silent Movies
- Automatically focus attention on objects in images
- Automatically answer questions about objects in a photograph
- Automatically turning sketches into photos
- Automatically create stylized images from rough sketches

These are all related to images … what else?

http://machinelearningmastery.com/inspirational-applications-deep-learning/

NOW (2016)

Uses of Artificial Neural Nets

- Automatic Handwriting Generation
- Character Text Generation
- Automatic Game Playing
- Automatic speech recognition
- Automatic speech understanding

- Explosion of companies
- Univ de Montreal: **$93,562,000**
  - Data Serving Canadians: Deep Learning and Optimization for the Knowledge Revolution

http://machinelearningmastery.com/inspirational-applications-deep-learning/

NOW (2016)

Uses of Artificial Neural Nets

- If you are NOT using Deep Nets for taskX, need to explain WHY!
  - Tried but inferior performance
  - Need to Explain response
  - Limited Training time

  - ? Ignorance ?

Neural Network Lore

- Neural nets have been adopted with an almost religious fervor within the AI community … several times
- Often ascribed near magical powers by people…
  - usually people who know the *least* about computation or brains ☺
- Circa 2005: For most AI people, magic was gone… but neural nets remained extremely interesting and useful mathematical objects
- Now: the magic is back!

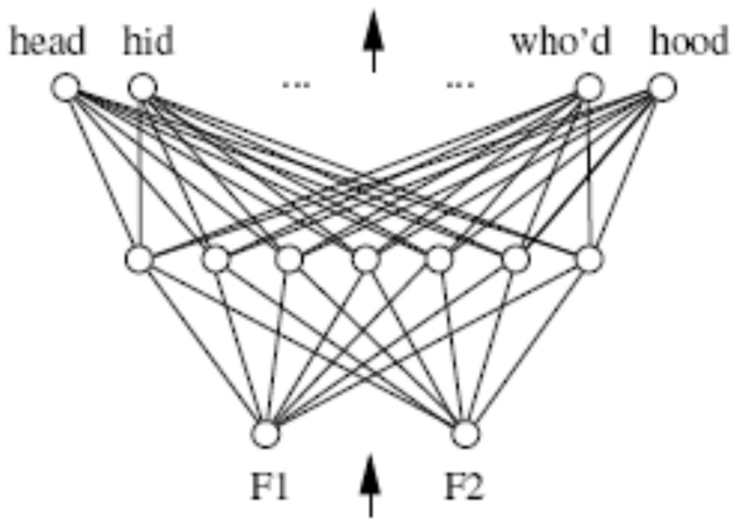
When to Consider Neural Networks

- Input is
  - high-dimensional (attribute-value pairs)
  - discrete or real-valued
  - possibly noisy [training, testing]
  - complete tuples
  - (eg, raw sensor input)
- Output is
  - vector of values
  - discrete or real valued
  - “linear ordering”

$\Rightarrow \Re^n \rightarrow \Re$

- ... have LOTS OF TIME to train
  (but performance is fast)
- Form of target function is unknown

- Human readability / Explanability is NOT important

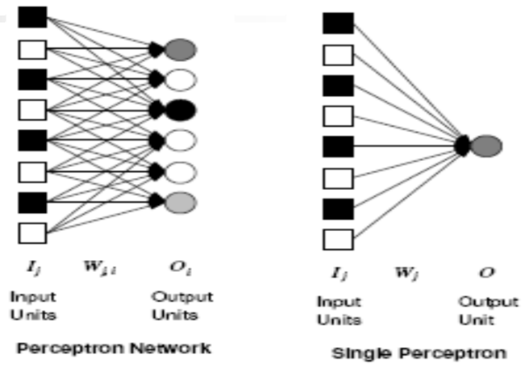


Reason for these Lectures…

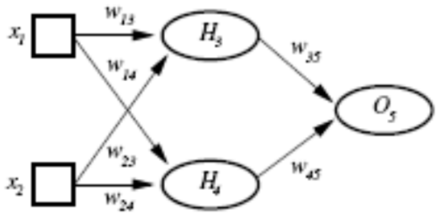
- Neural nets are really used
  - Deep nets seems to work REALLY well
  - … but still kinda ad hoc
- Main reason:
  to introduce some foundational ideas
  - gradient descent … for non-trivial function
  - tricks: conjugate gradient, etc.

Jump

Types of Network Structures



- Single layer:
  - Linear Threshold Units
  - Linear Units, Sigmoid Units

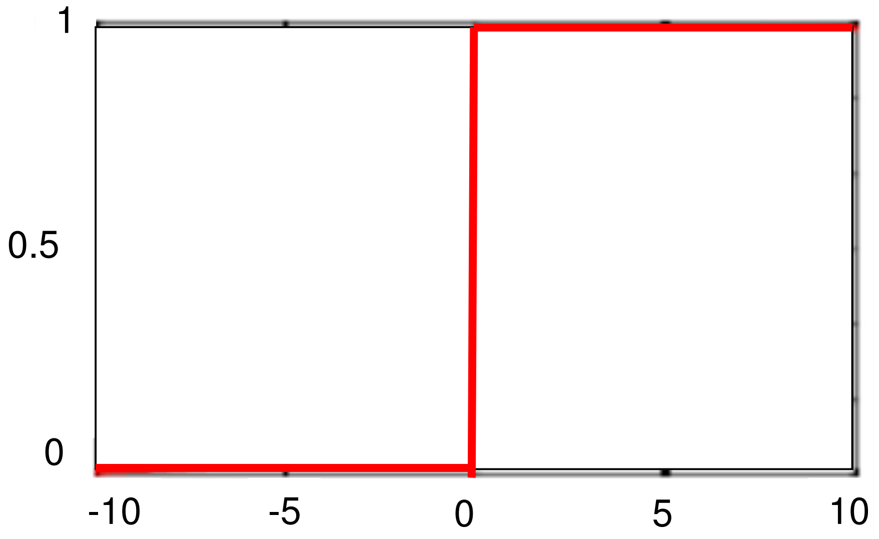


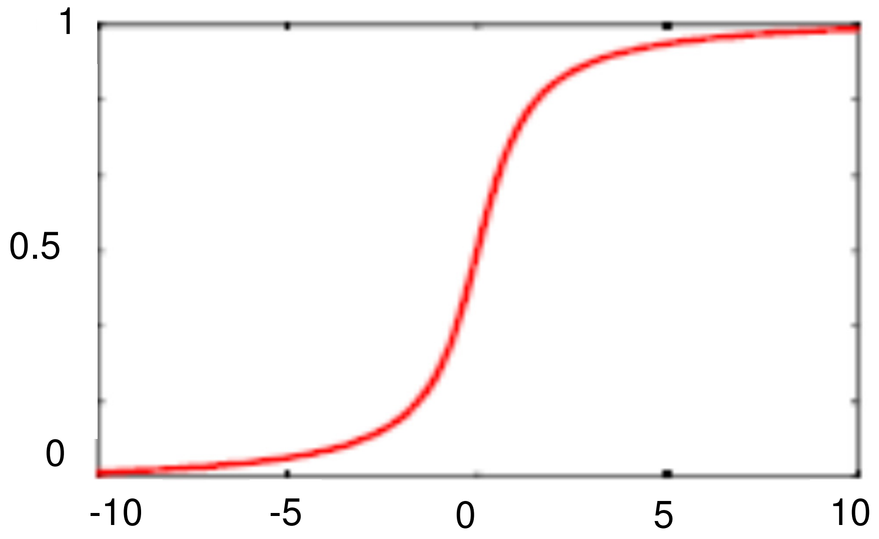
- (Shallow) multi-layered feed-forward:
  - input → hidden units → output

- Deep multi-layered feed-forward

- Recurrent + Cycles, to allow "state"
  - Hopfield networks (used for associative memory), Boltzmann machines,...

Threshold Functions

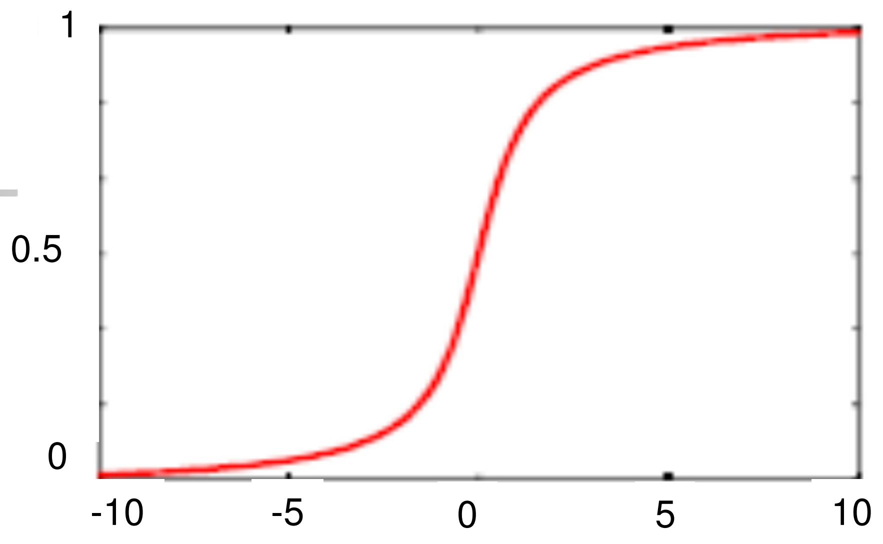




g(x) ≈ sign(x)
    = 2 * sign(x) – 1
(perceptron; step; Heaviside)

g(x) = $\frac{1}{1+\exp(-x)}$
(logistic regression; sigmoid)…
or tanh(x)

Sigmoid Unit



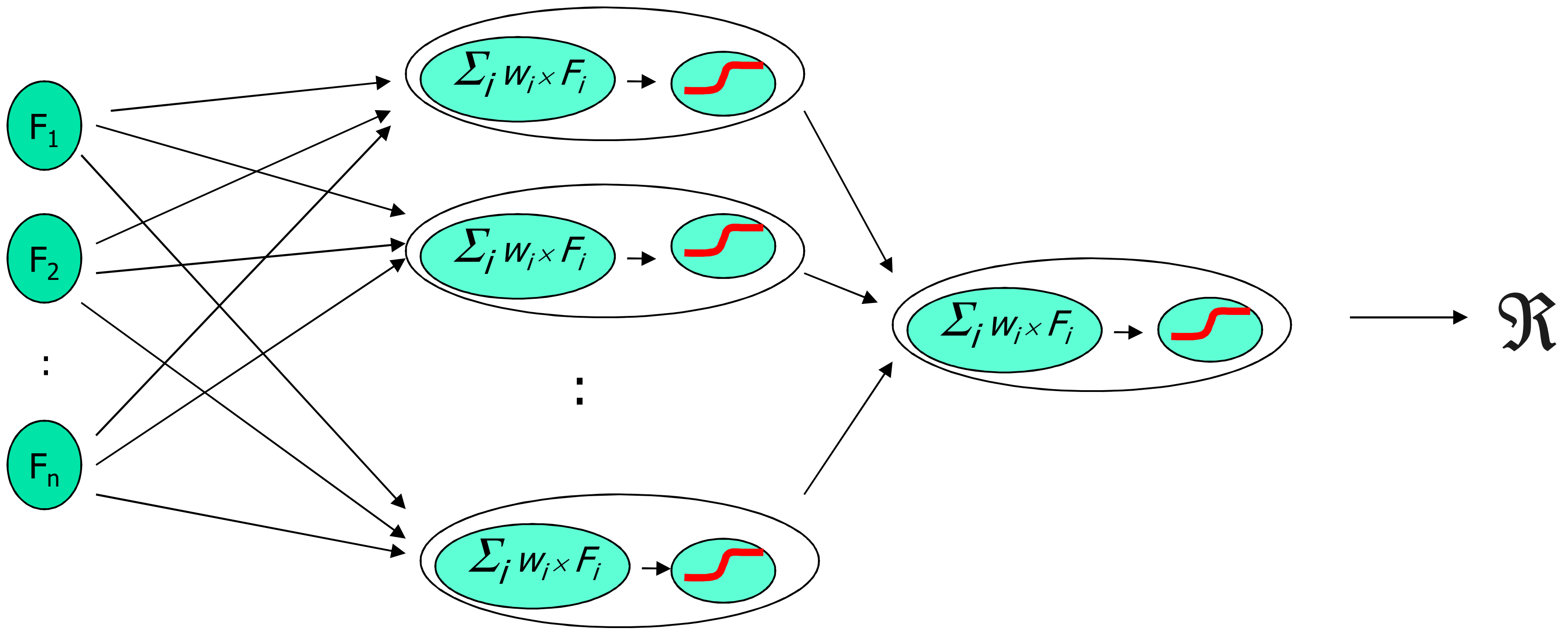
- Sigmoid function:

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

- Useful properties
  - $\sigma : \Re \rightarrow [0,1]$
  - If $x \approx 0$, then $\sigma(x) \approx x + \frac{1}{2}$
  - $\frac{\partial\, \sigma(x)}{\partial x} = \sigma(x)\ (1 - \sigma(x))$

Feed Forward Neural Nets

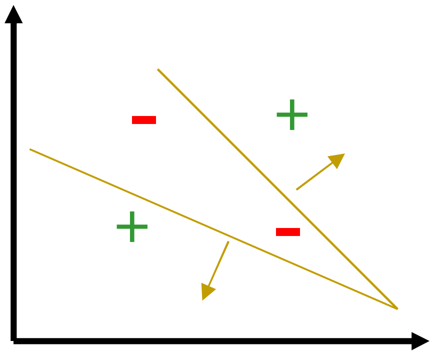
- *SET* of **connected Sigmoid Functions**

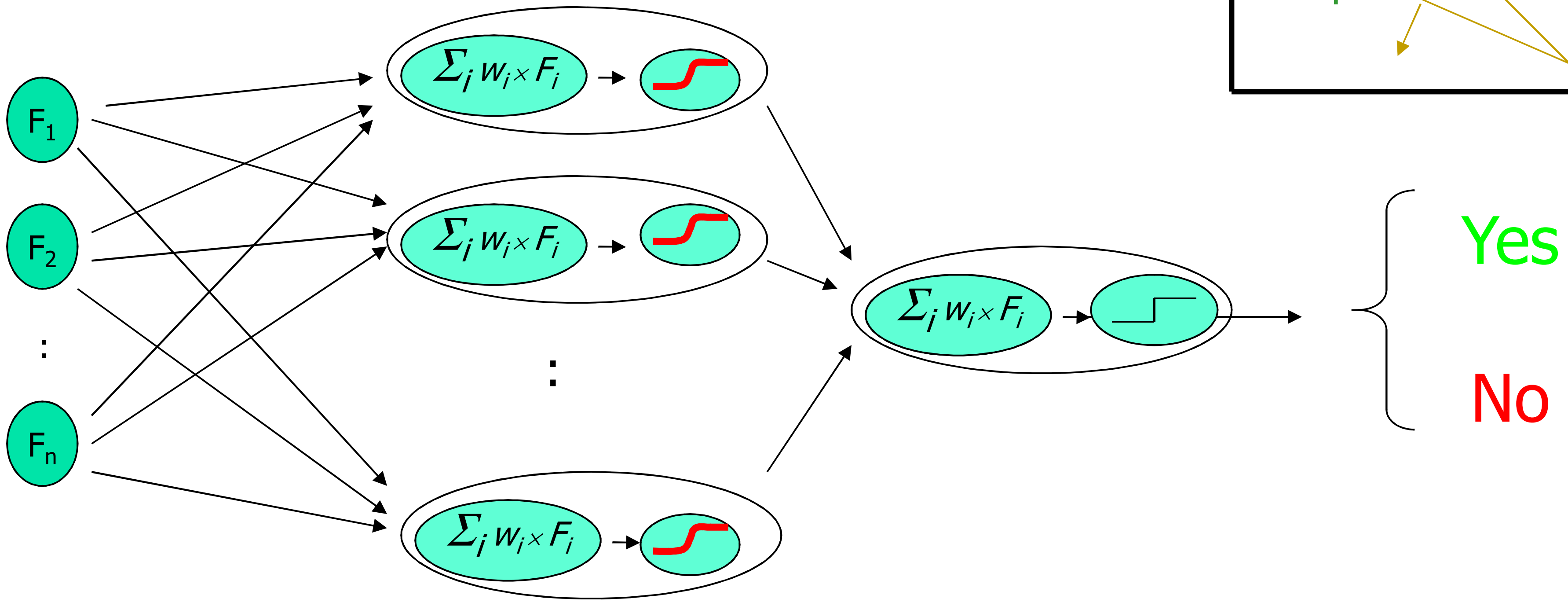


F_1
F_2
:
F_n
$\Sigma_i w_i \times F_i$
$\Sigma_i w_i \times F_i$
:
$\Sigma_i w_i \times F_i$
$\Sigma_i w_i \times F_i$
$\Re$

Artificial Neural Nets

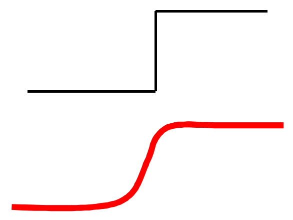
- Can Represent ≈*ANY* classifier!
  - w/just 1 "hidden" layer…

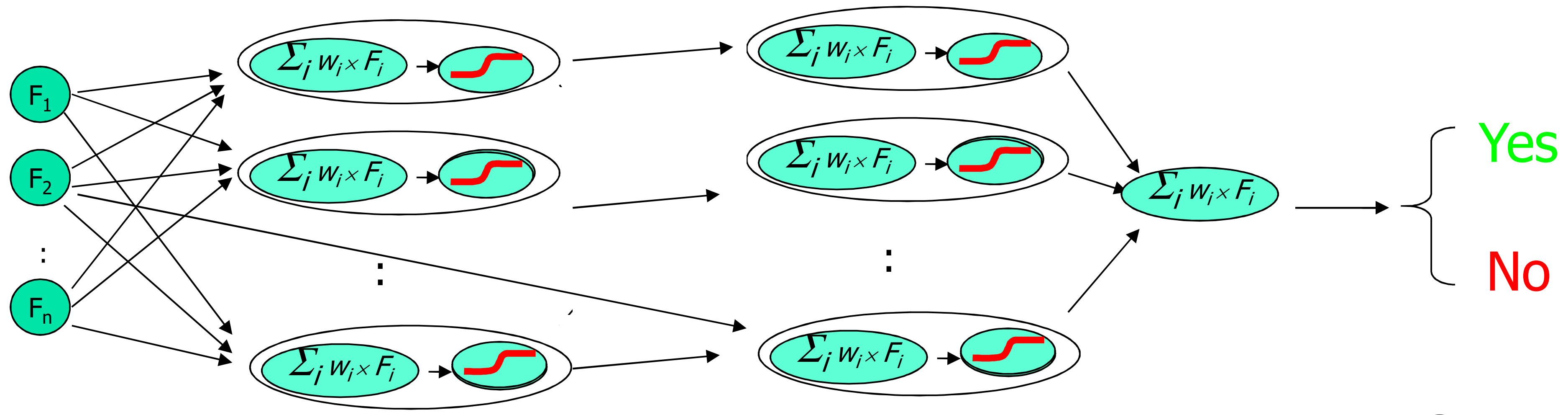



F_1
F_2
:
F_n
$\Sigma_i w_i \times F_i$
$\Sigma_i w_i \times F_i$
:
$\Sigma_i w_i \times F_i$
$\Sigma_i w_i \times F_i$
Yes
No

ANNs: Architecture

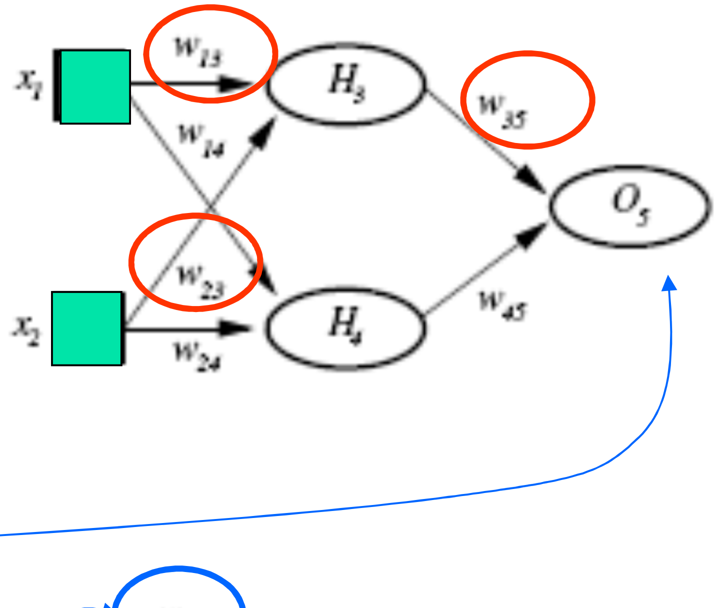
- Different # of layers
- Different structures
    - what's connected to what..
- Different "squashing function"



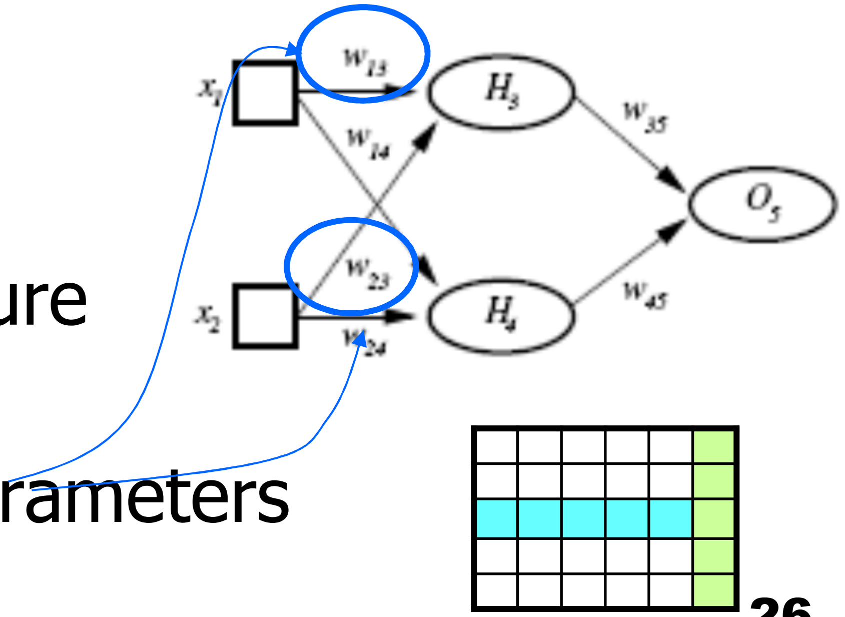


Tasks

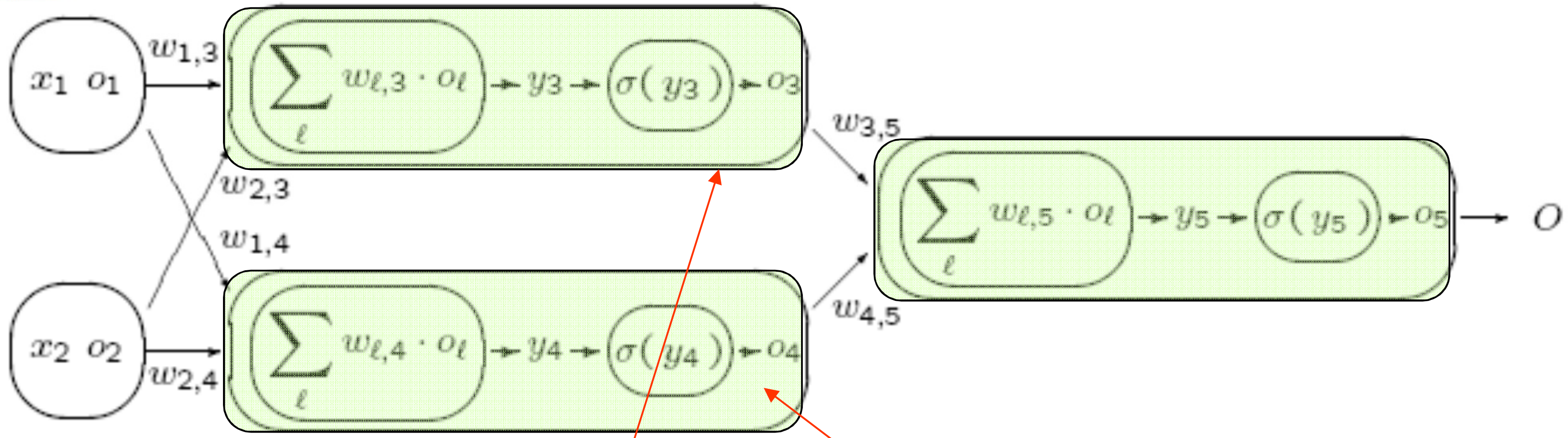
- Computing Response
  - Given Network Structure
    - + Parameters
    - + Instance
  - Compute response



- Training
  - Given Network Structure
    - + Data Sample
  - Compute values of Parameters



Computing Network Output



x_1 o_1
x_2 o_2
$w_{1,3}$
$w_{2,3}$
$w_{1,4}$
$w_{2,4}$
$\sum_\ell w_{\ell,3} \cdot o_\ell \to y_3 \to \sigma(y_3) \to o_3$
$\sum_\ell w_{\ell,4} \cdot o_\ell \to y_4 \to \sigma(y_4) \to o_4$
$w_{3,5}$
$w_{4,5}$
$\sum_\ell w_{\ell,5} \cdot o_\ell \to y_5 \to \sigma(y_5) \to o_5 \to O$

- Two (non-input) layers: 2 input units + 2 hidden units + 1 output unit
- "Activation" passed from input to output:

$$\begin{aligned} o &= \sigma\left(\textstyle\sum_r w_{r,5} \cdot o_r \right) = \sigma\left(w_{3,5} \cdot o_3 + w_{4,5} \cdot o_4 \right) \\ &= \sigma\left(w_{3,5} \cdot \boxed{\sigma(\textstyle\sum_s w_{s,3} \cdot o_s)} + w_{4,5} \cdot \boxed{\sigma(\textstyle\sum_t w_{t,4} \cdot o_t)} \right) \\ &= \sigma\big(\quad w_{3,5} \cdot \sigma(w_{1,3} \cdot o_1 + w_{2,3} \cdot o_2) \\ &\qquad + w_{4,5} \cdot \sigma(w_{1,4} \cdot o_1 + w_{2,4} \cdot o_2) \big) \end{aligned}$$

Node #0 set to "1" is input to each node (using $w_{0,t}$)
Final unit (here "#5") is $\sigma(\cdot)$ here… but often just the y_5 …

Representational Power

- **Any Boolean Formula**
  - Consider formula in DNF: $(x_1 \,\&\, \neg x_2) \vee (x_2 \,\&\, x_4) \vee (\neg x_3 \,\&\, x_5)$
  - Represent each AND by hidden unit; the OR by output unit
  - ... but may need exponentially-many hidden units!

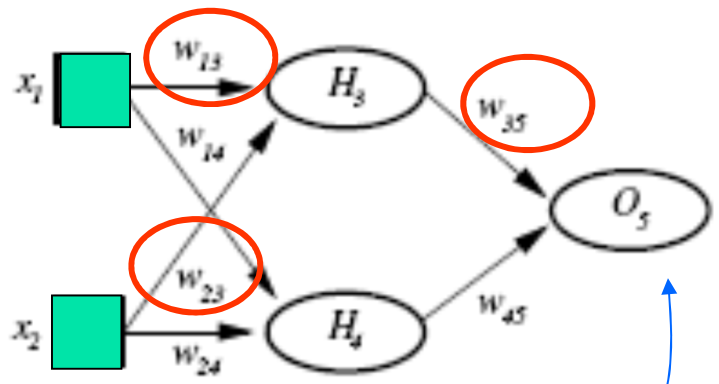
- **Bounded functions**
  - Can approximate *any bounded continuous function* to *arbitrary accuracy* with **1** hidden sigmoid layer
    + linear output unit
  - ... given enough hidden units!

  (Output unit "linear" $\Rightarrow$ computes $\hat{y} = W_4 \cdot A$)

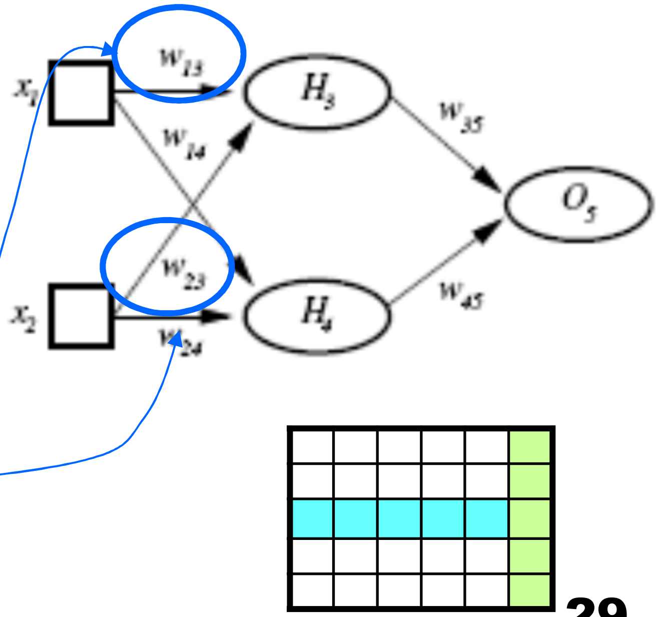
- **Arbitrary Functions**
  - Can approximate *any function* to *arbitrary accuracy* with **2** hidden sigmoid layers + linear output unit

Tasks

- Computing Response
  - Given Network Structure
    + Parameters
    + Instance
  - Compute response



- Training
  - Given Network Structure
    + Data Sample
  - Compute Parameters



MultiLayerNetwork Learning Task

- Want to minimize error on training examples
  [not quite... why?]

$\Rightarrow$ function minimization problem

$$\boxed{Err_D(\mathbf{w}) = \frac{1}{2}\sum_{[\mathbf{x},t]\in D}(t - o_{\mathbf{w}}(\mathbf{x}))^2}$$

Err on *D* (outputs **t**, for given inputs **x**),
is function of weights **w** = { w_{ij} }

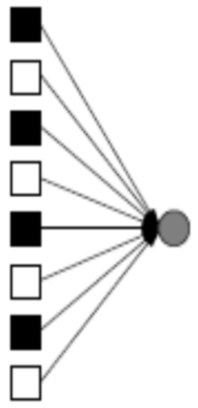
- Minimize using
  gradient descent in weight space:

  $\Rightarrow$ backpropagation algorithm (aka "chain rule")

Need for Backpropagation

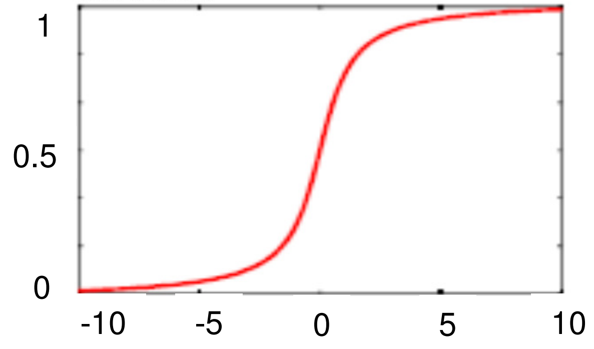
- Perceptron learning relied on direct connection between input value x_j, weight w_j, output value t
  $\Rightarrow$ could localize contribution & determine change locally

- Not true for multilayer network!



- Still, can estimate effect of each weight
  ... and make small changes accordingly
  - Use derivative of error, wrt weight w_{ij} !
  - Propagate backward (up net) using chain rule

  but no guarantees here, as $\exists$ many local minima!

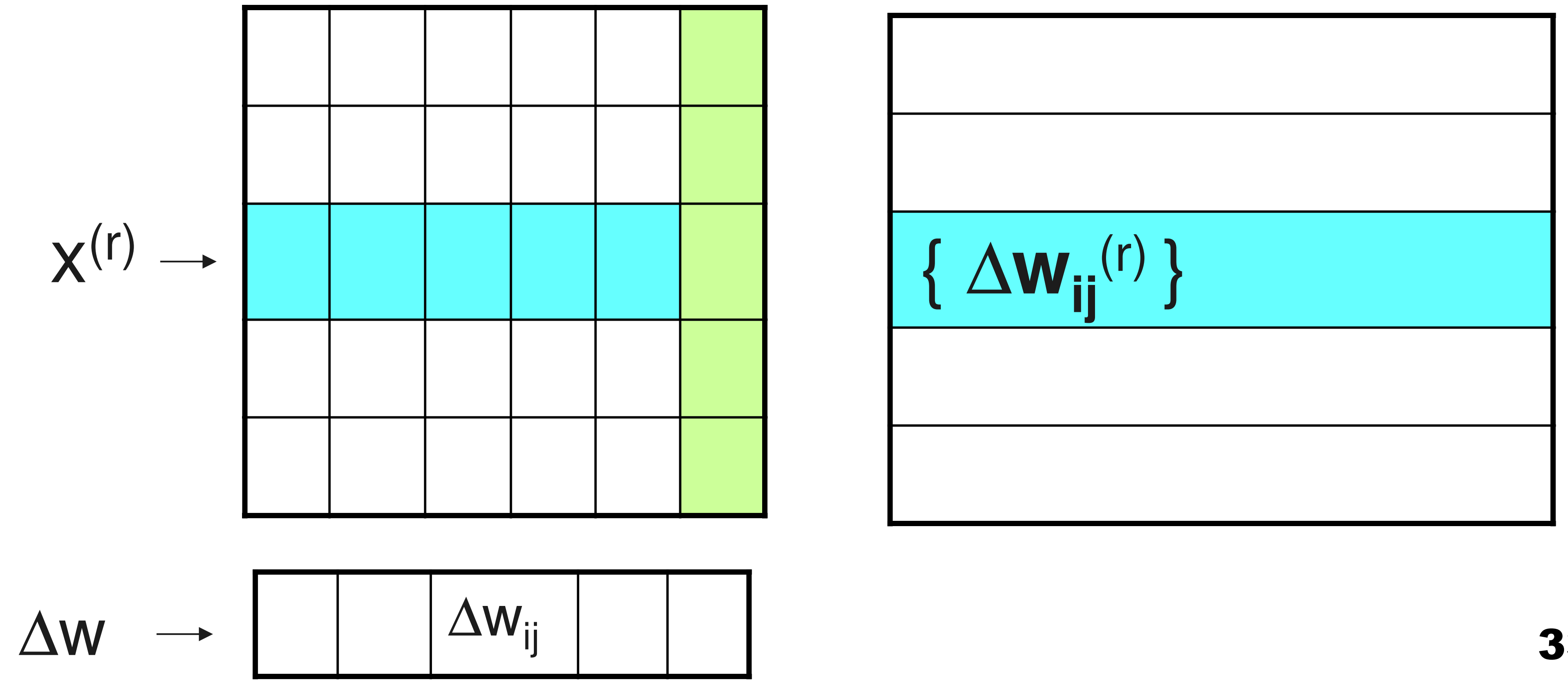


- Need to take DERIVATIVE
  $\Rightarrow$ use "sigmoid" squashing function...

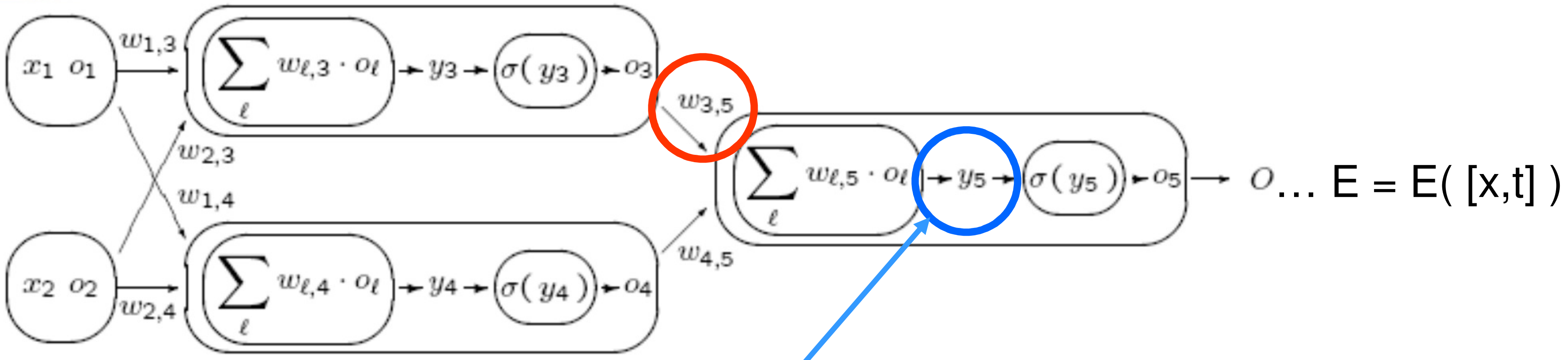
Generic Gradient Descent

```
0. New $\mathbf{w}$
    $\Delta\mathbf{w} := 0$
1. For each instance $r$, compute
    $\Delta\mathbf{w}_{ij}^{(r)} := \ldots$

    $\Delta\mathbf{w}_{ij} \mathrel{+}= \Delta\mathbf{w}_{ij}^{(r)}$
2. Increment $w \mathrel{+}= \eta\, \Delta\mathbf{w}$
```

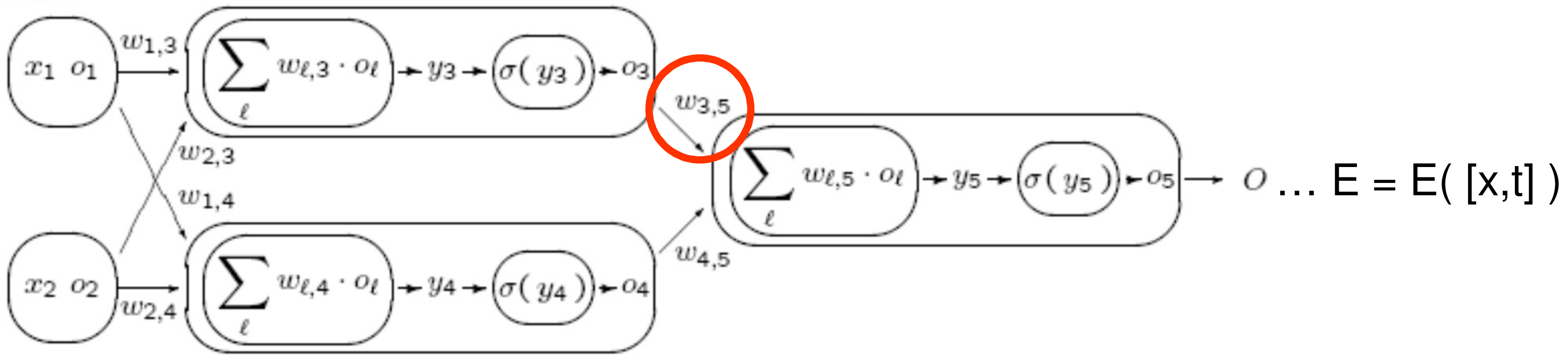


Error Gradient for Network



- E = E([**x**, t]) = $\frac{1}{2}$(O$_{\mathbf{w}}$(**x**) – t)^2
- Let $y_i = \sum_r w_{r,i}\, o_r \qquad \delta_i \triangleq \frac{\partial E}{\partial y_i}$
- $\frac{\partial E(\,[\mathbf{x}, t]\,)}{\partial w_{3,5}} = \frac{\partial E}{\partial y_5}\frac{\partial y_5}{\partial w_{3,5}} = \delta_5 \frac{\partial y_5}{\partial w_{3,5}}$
- $\frac{\partial y_5}{\partial w_{3,5}} = \frac{\partial\left(\sum_r w_{r,5}\, o_r\right)}{\partial w_{3,5}} = \frac{\partial\left(w_{3,5}\, o_3 + w_{4,5}\, o_4\right)}{\partial w_{3,5}} = o_3$
- $\Rightarrow \boxed{\frac{\partial E(\,[\mathbf{x},t]\,)}{\partial w_{3,5}} = \delta_5\, o_3}$

Factoring Derivative

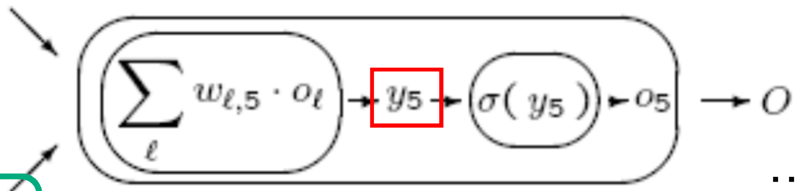


- Here $\boxed{\dfrac{\partial E([\mathbf{x}, t])}{\partial w_{3,5}} = \delta_5 o_3}$
- In general: $\dfrac{\partial E([\mathbf{x}, t])}{\partial w_{i,j}} = \dfrac{\partial E}{\partial y_j} \dfrac{y_j}{\partial w_{i,j}} = \delta_j o_i$

$$\boxed{\frac{\partial E([\mathbf{x}, t])}{\partial w_{i,j}} = \delta_j \, o_i}$$

- Compute each o_i during FORWARD sweep
- Compute each δ_j during BACKWARD sweep

Computing "Terminal" δ_is



… E = E([x,t])

… computed output is o

… should be t

- $\delta_5 = \frac{\partial E}{\partial y_5} = \boxed{\frac{\partial E}{\partial o_5}} \boxed{\frac{\partial o_5}{\partial y_5}}$

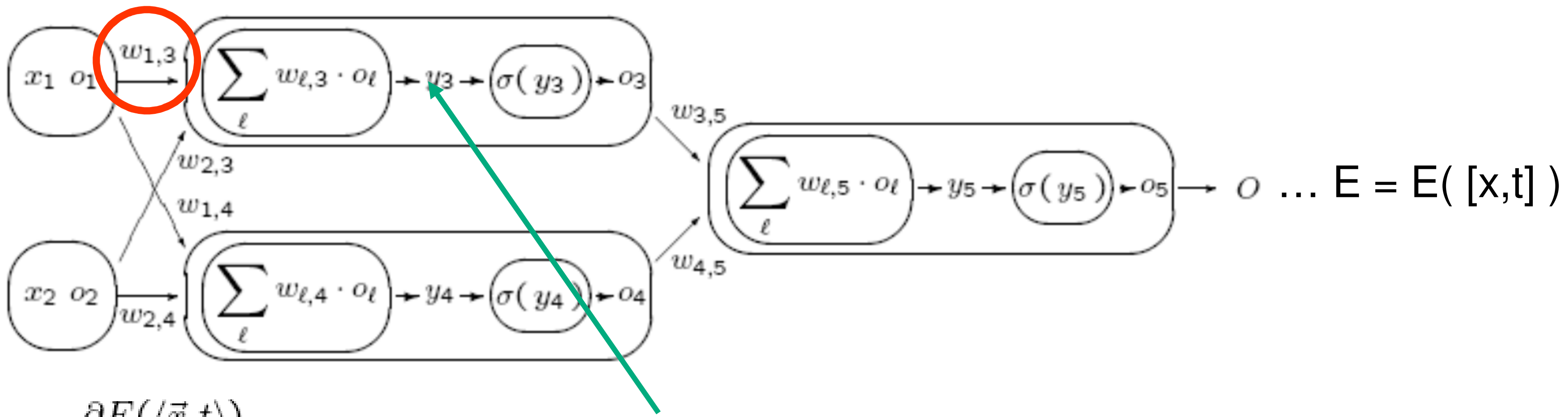
- $\boxed{\frac{\partial E([\mathbf{x},t])}{\partial o_5}} = \frac{\partial}{\partial o_5}\left[\frac{1}{2}(o_5 - t)^2\right] = (\mathrm{o}_5 - \mathrm{t})\frac{\partial}{\partial o_5}(\mathrm{o}_5 - \mathrm{t}) = (\mathrm{o}_5 - \mathrm{t})$

- $\boxed{\frac{\partial o_5}{\partial y_5}} = \frac{\partial \sigma(y_5)}{\partial y_5} = \sigma(y_5)\big(1 - \sigma(y_5)\big) = o_5(1 - o_5)$

$$\Rightarrow \boxed{\delta_5 = (o_5 - t)\ o_5\ (1 - o_5)}$$

- Familiar … just like LMS for 1-layer (0 hidden units) !

Computing Non-Terminal δ_is

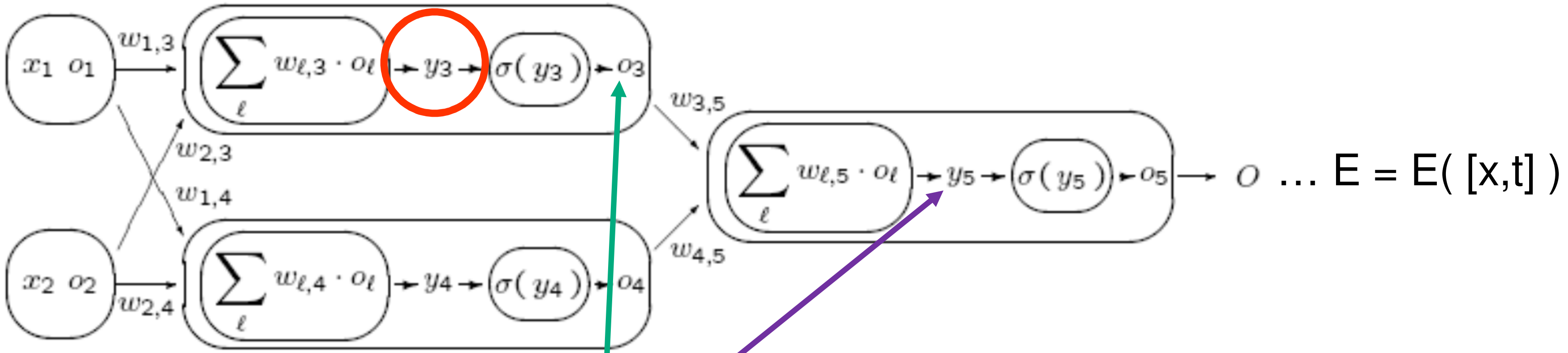


As $\frac{\partial E(\langle \vec{x}, t \rangle)}{\partial w_{1,3}}$ depends only on y_3

$$\Rightarrow \boxed{\frac{\partial E(\langle \vec{x}, t \rangle)}{\partial w_{1,3}} = \frac{\partial E}{\partial y_3} \frac{\partial y_3}{\partial w_{1,3}} = \delta_3 \; o_1}$$

- $\frac{\partial y_3}{\partial w_{1,3}} = \frac{\partial(\sum_\ell w_{\ell,3} o_\ell)}{\partial w_{1,3}} = o_1$
- $\delta_3 = \frac{\partial E}{\partial y_3} = \boxed{\;\dots\;}$

Computing δ_3

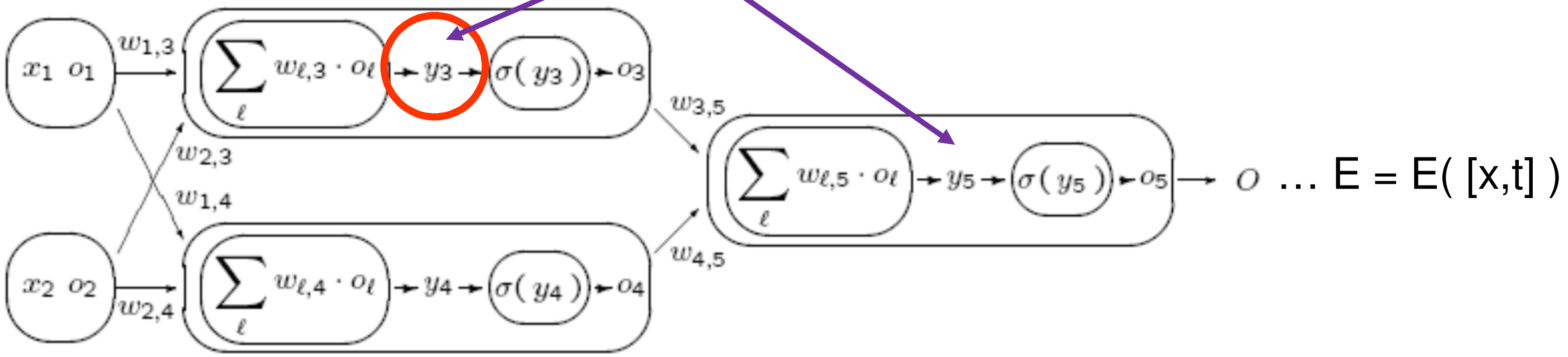


- $\delta_3 = \frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial o_3} \boxed{\frac{\partial o_3}{\partial y_3}}$
- $\frac{\partial E}{\partial o_3} = \frac{\partial E}{\partial y_5}\frac{\partial y_5}{\partial o_3} = \delta_5 \frac{\partial(\sum_r w_{r,5} o_r)}{\partial o_3} = \delta_5 \; w_{3,5}$
- $\boxed{\frac{\partial o_3}{\partial y_3}} = \frac{\partial \sigma(y_3)}{\partial y_3} = \sigma(y_3)\,(1 - \sigma(y_3)\,) = o_3(1 - o_3)$

$\Rightarrow \quad \delta_3 = \boxed{[\delta_5 \; w_{3,5}]\,[o_3(1 - o_3)]}$

So once we know δ_5, it is easy compute δ_3

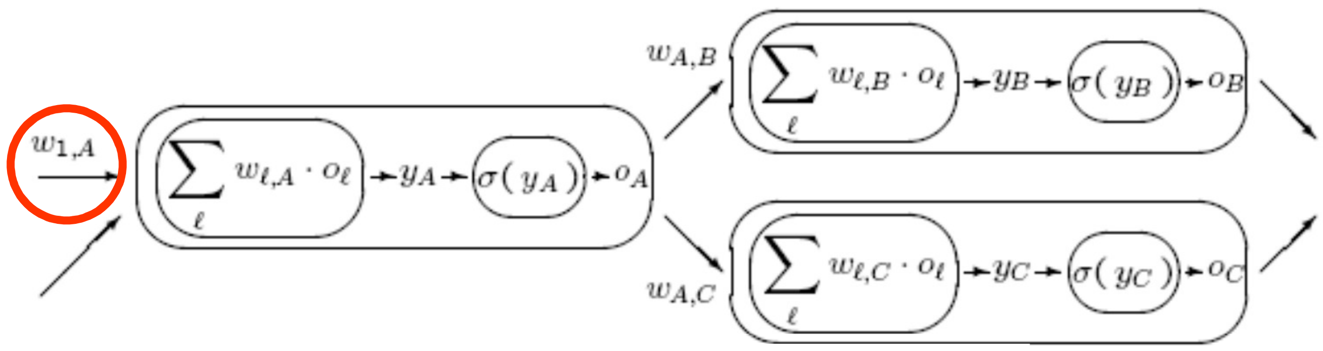
… Going BACKWARDS!



- $\delta_3 = \frac{\partial E}{\partial y_3} = \frac{\partial E}{\partial o_3} \frac{\partial o_3}{\partial y_3}$
- $\frac{\partial E}{\partial o_3} = \frac{\partial E}{\partial y_5} \frac{\partial y_5}{\partial o_3} = \delta_5 \frac{\partial \left(\sum_r w_{r,5} o_r\right)}{\partial o_3} = \delta_5 \; w_{3,5}$
- $\frac{\partial o_3}{\partial y_3} = \frac{\partial \sigma(y_3)}{\partial y_3} = \sigma(y_3)\,(1 - \sigma(y_3)\,) = o_3(1 - o_3)$

$\Rightarrow \quad \delta_3 = \boxed{[\delta_5 \; w_{3,5}]\,[o_3(1 - o_3)]}$

What if Many Children?



$w_{1,A}$
$\sum_\ell w_{\ell,A} \cdot o_\ell$
y_A
$\sigma(y_A)$
o_A
$w_{A,B}$
$\sum_\ell w_{\ell,B} \cdot o_\ell$
y_B
$\sigma(y_B)$
o_B
$w_{A,C}$
$\sum_\ell w_{\ell,C} \cdot o_\ell$
y_C
$\sigma(y_C)$
o_C

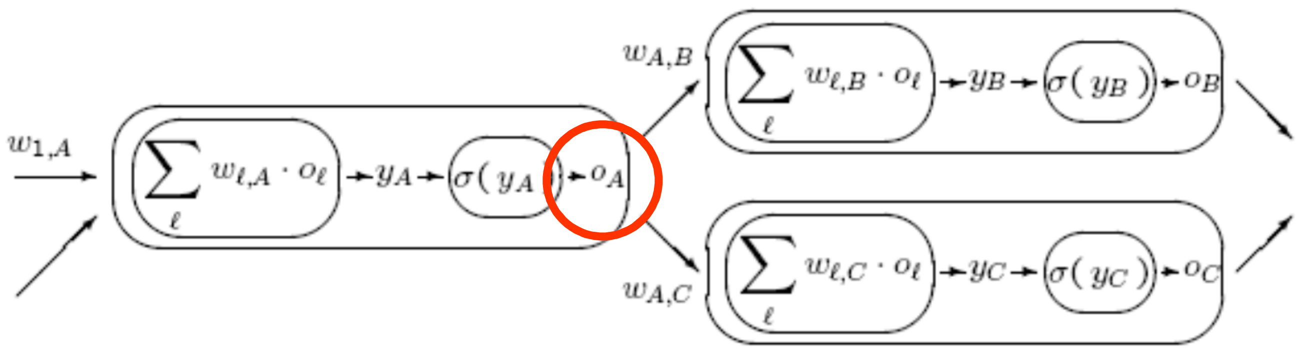
- As before. . .

$$\frac{\partial E}{\partial w_{1,A}} = \frac{\partial E}{\partial y_A} \frac{\partial y_A}{\partial w_{1,A}} = \delta_A \, o_A$$

$$\delta_A = \frac{\partial E}{\partial y_A} = \frac{\partial E}{\partial o_A} \frac{\partial o_A}{\partial y_A} = \frac{\partial E}{\partial o_A} [o_A(1 - o_A)]$$

- Notice $\frac{\partial E}{\partial o_A}$ depends only on BOTH
  - ⋆ B (via y_B)
  - ⋆ C (via y_C)

Multiple Children (con't)



$w_{1,A}$
$\sum_\ell w_{\ell,A} \cdot o_\ell$
y_A
$\sigma(y_A)$
o_A
$w_{A,B}$
$\sum_\ell w_{\ell,B} \cdot o_\ell$
y_B
$\sigma(y_B)$
o_B
$w_{A,C}$
$\sum_\ell w_{\ell,C} \cdot o_\ell$
y_C
$\sigma(y_C)$
o_C

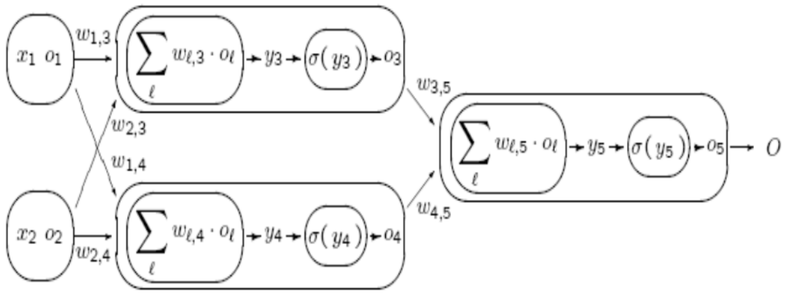
- $$\frac{\partial E}{\partial o_A} = \frac{\partial E}{\partial y_B}\frac{\partial y_B}{\partial o_A} + \frac{\partial E}{\partial y_C}\frac{\partial y_C}{\partial o_A} = \sum_{k \in child(A)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial o_A}$$

$$= \sum_{k \in child(A)} \delta_k \frac{\partial(\sum_\ell w_{\ell,k} \cdot o_\ell)}{\partial o_A} = \sum_{k \in child(A)} \delta_k w_{A,k}$$

Here: $\delta_A = o_A(1 - o_A)\left[\delta_B w_{A,B} + \delta_C w_{A,C}\right]$

- In general: $\boxed{\delta_\ell = o_\ell(1 - o_\ell) \sum_{k \in child(\ell)} \delta_k w_{\ell,k}}$

Backpropagation

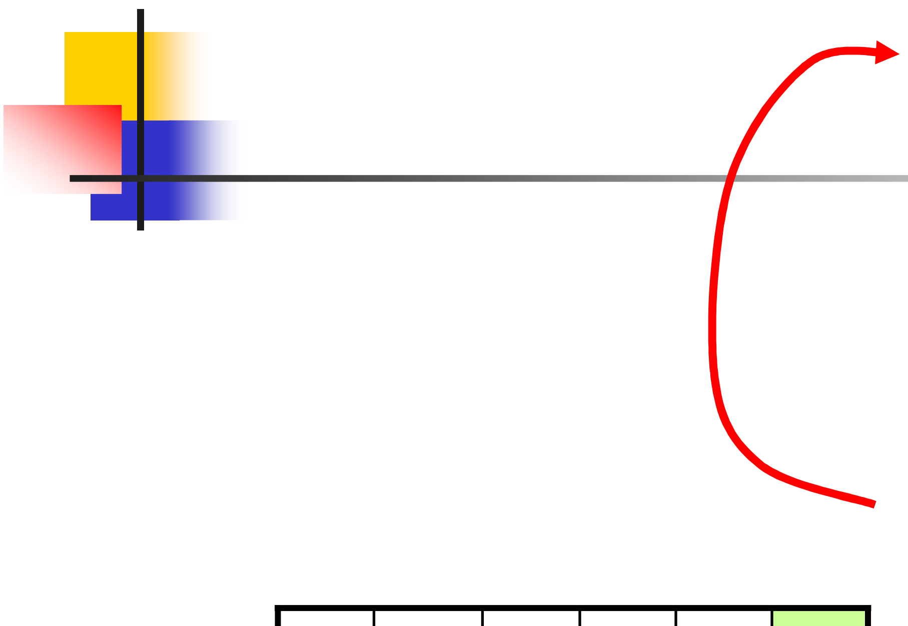


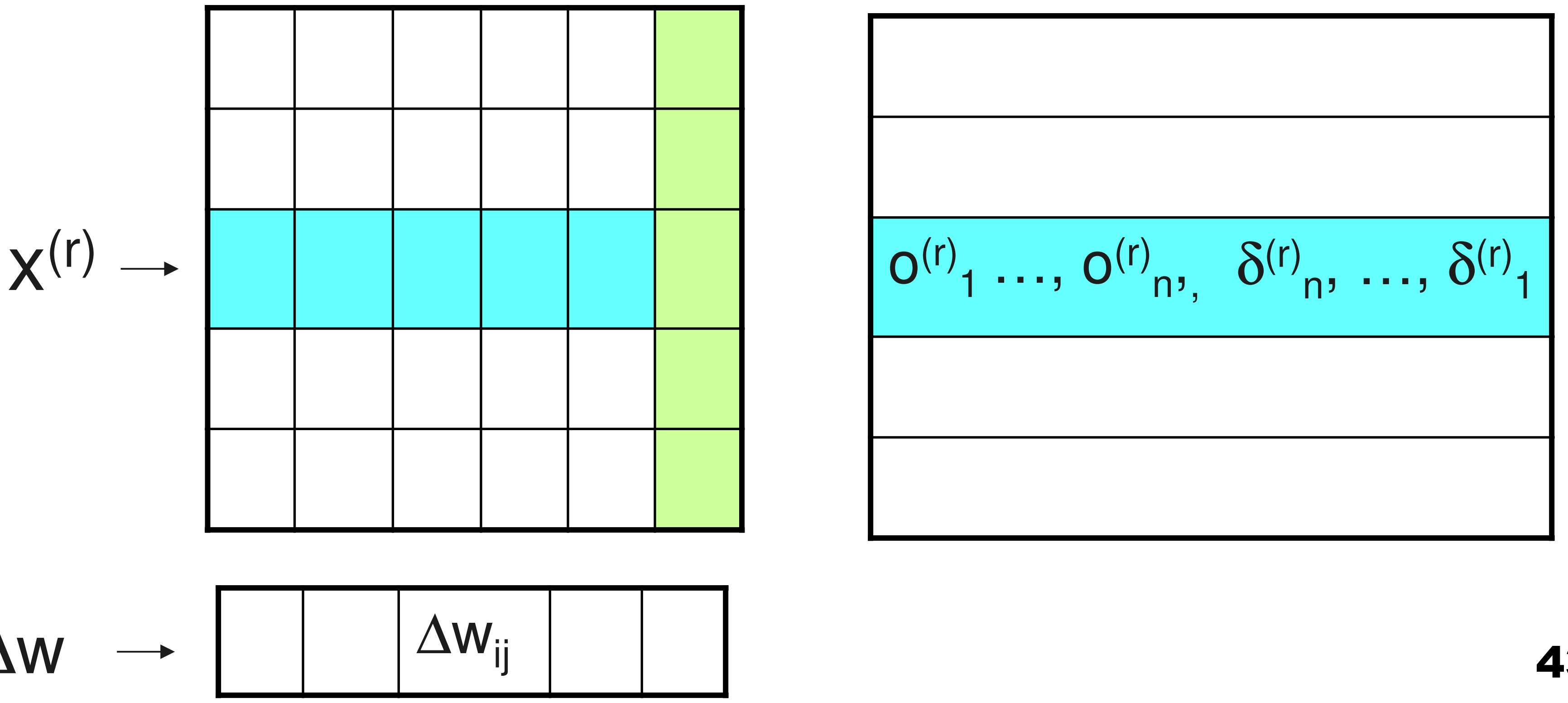
```
Initialize all weights to small random numbers
Until satisfied, do
  ▪ For each training example [x, t] , do
      1. Sweep forward
           Compute network outputs $o_k$ for x for each hidden/output node
      2. Sweep backward
           For each output unit k
               $\delta_k \leftarrow o_k (1 - o_k) (t_k - o_k)$
           For each hidden unit h
               $\delta_h \leftarrow o_h (1 - o_h) \sum_{k \in \text{child}(h)} w_{h,k} \delta_k$
      3. Update each network weight
               $w_{i,j} \leftarrow w_{i,j} + \eta \delta_j o_i$
```

Notice everything is trivial to compute!

Skip

0. New **w**
   Δ**w** := 0
1. For each instance r, compute
   a. Forward: $o^{(r)}_i := \sigma(\sum_j w_{ji}\, o^{(r)}_j)$
   b. Backward: $\delta_i^{(r)} = \frac{\partial E^{(r)}}{\partial y^{(r)}}$
   c. $\Delta \mathbf{w}_{ij}$ += $\delta^{(r)}_j\, o^{(r)}_i$
2. Increment w += $\eta\, \Delta$**w**



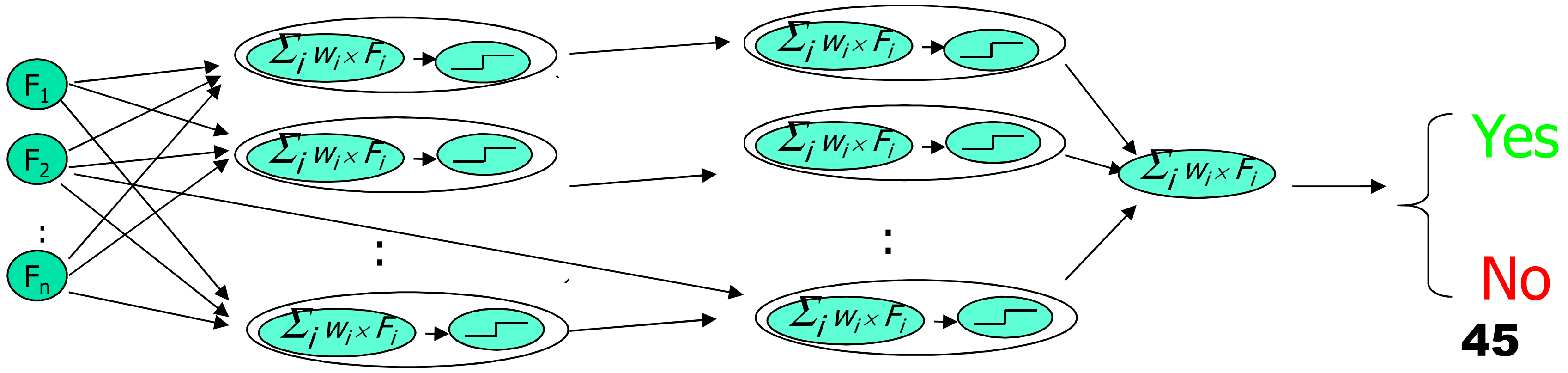

$x^{(r)}$
$o^{(r)}_1 \ldots, o^{(r)}_n, \ \delta^{(r)}_n, \ldots, \delta^{(r)}_1$
Δw
Δw_{ij}

More on Backpropagation

- Gradient descent over entire network weight vector **w**= [w_{ij}]
- Can be either: "Incremental Mode" Gradient Descent or "Batch Mode":

$$\frac{\partial E}{\partial w_i} = \sum_{[x,t]\in D} \frac{\partial E^{([x,t])}}{\partial w_i}$$

- Easily generalized to arbitrary directed graphs
  - If have > 1 OUTPUTs: Just add them up!
  - Can have arbitrary connections
    Not just "everything on level 3 to everything on level 4"



Efficiency

- **Learning**: Intractable in general
  - Training can take thousands of iterations... slow!
  - Learning net w/ single hidden unit is NP-hard
  - In practice: backprop is very useful

- **Use**: Using network (after training) is very fast

Issues

Backprop will (at best)…

- … slowly …
  - Conjugate gradient, …
  - Line search
- … converge to LOCAL Opt …
  - Multiple restart
  - simulated annealing, …
- … wrt Training Data
  - Early stopping
  - regularization, …