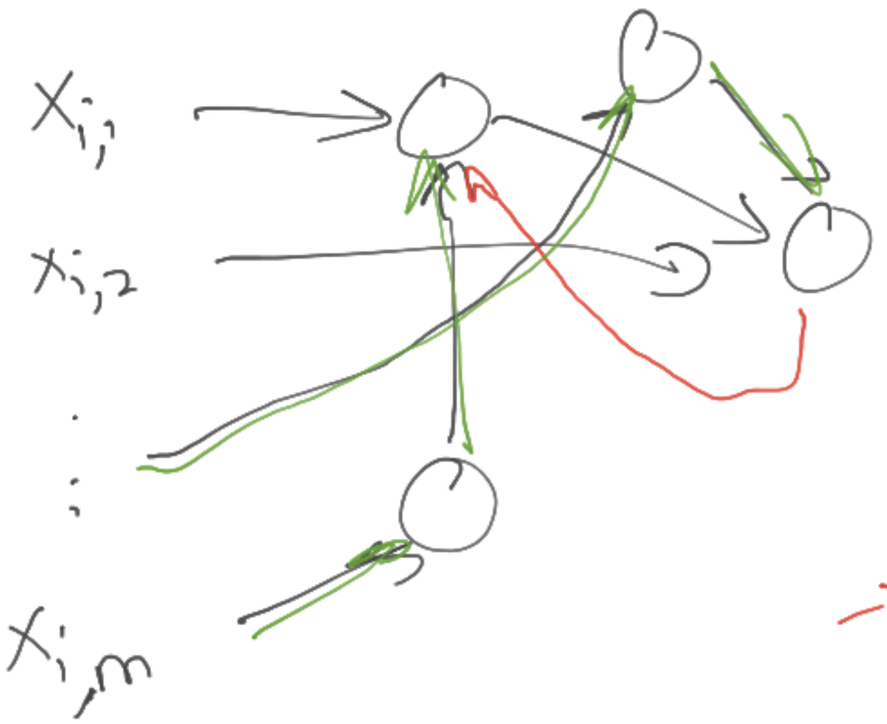
Artificial Neural Networks (ANN)

Lecture 8
-ANNs!



$\rightarrow f_{w_k}(x_i)$

"Network architecture" is how perceptrons are wired together. (and how many).

→ recurrent network: when the graph is cyclic.

- We consider non-recurrent networks.

○ → perceptron
→ unit
→ "node"
→ vertex.

Perceptron

$$(x_i, y_i)_{i=1}^{n}$$

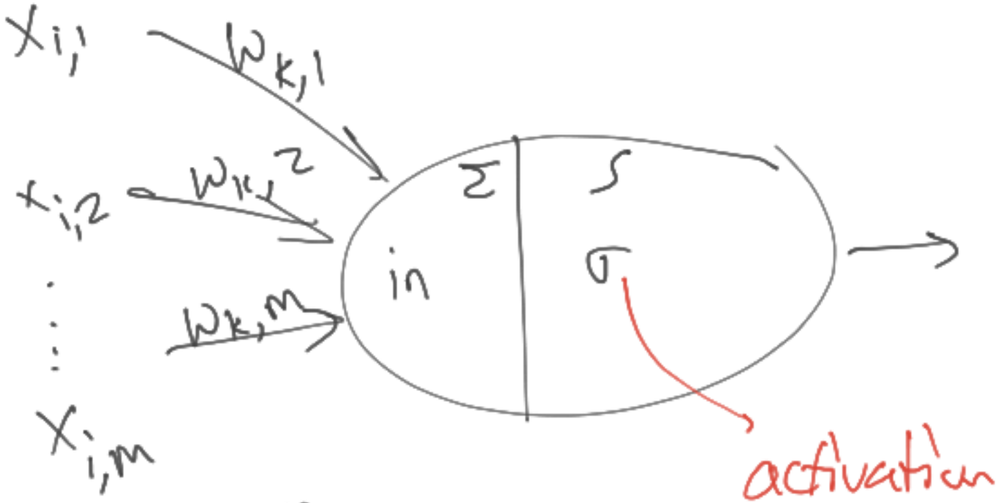
$$\ell(w_k) = \sum_{i=1}^{n} (y_i - f_{w_k}(x_i))^2$$

$$\forall j : w_{k+1,j} := w_{k,j} - \alpha \frac{\partial \ell(w_k)}{\partial w_{k,j}}$$

$$\frac{\partial \ell(w_k)}{\partial w_{k,j}} = \sum_i (y_i - f_{w_k}(x_i)) \frac{\partial f_{w_k}(x_i)}{\partial w_{k,j}}$$

new notation:

$$\frac{\partial f_w(x)}{\partial w_{i,j}}$$



activation function.

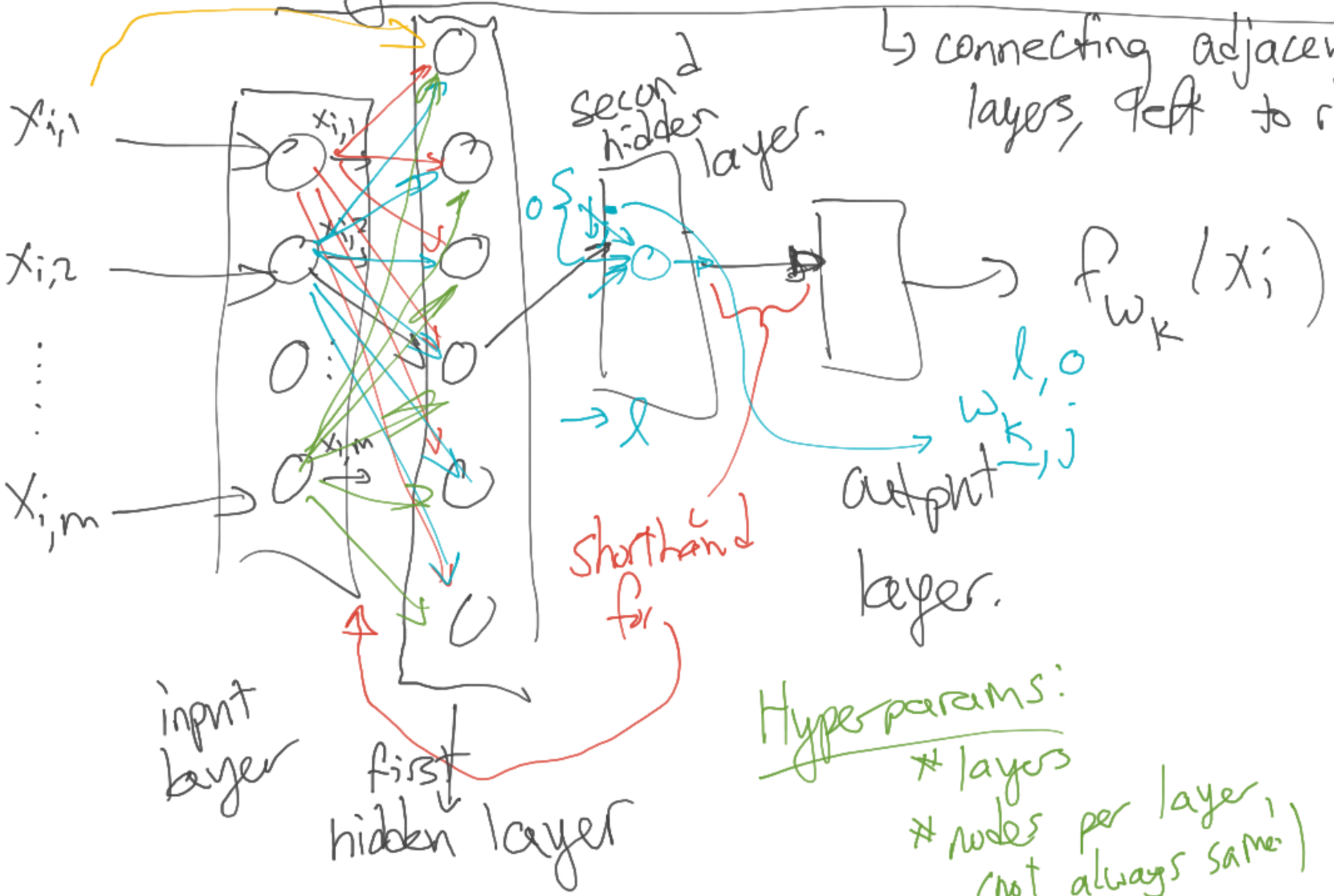
"nonlinearity"

$$in = \sum_{j=1}^{m} x_{i,j} \, w_{k,j}$$

$$f_{w_k}(x_i) = \sigma(in)$$

Fully Connected Feedforward Network.

↳ connecting adjacent layers, left to right.



Hyperparams:

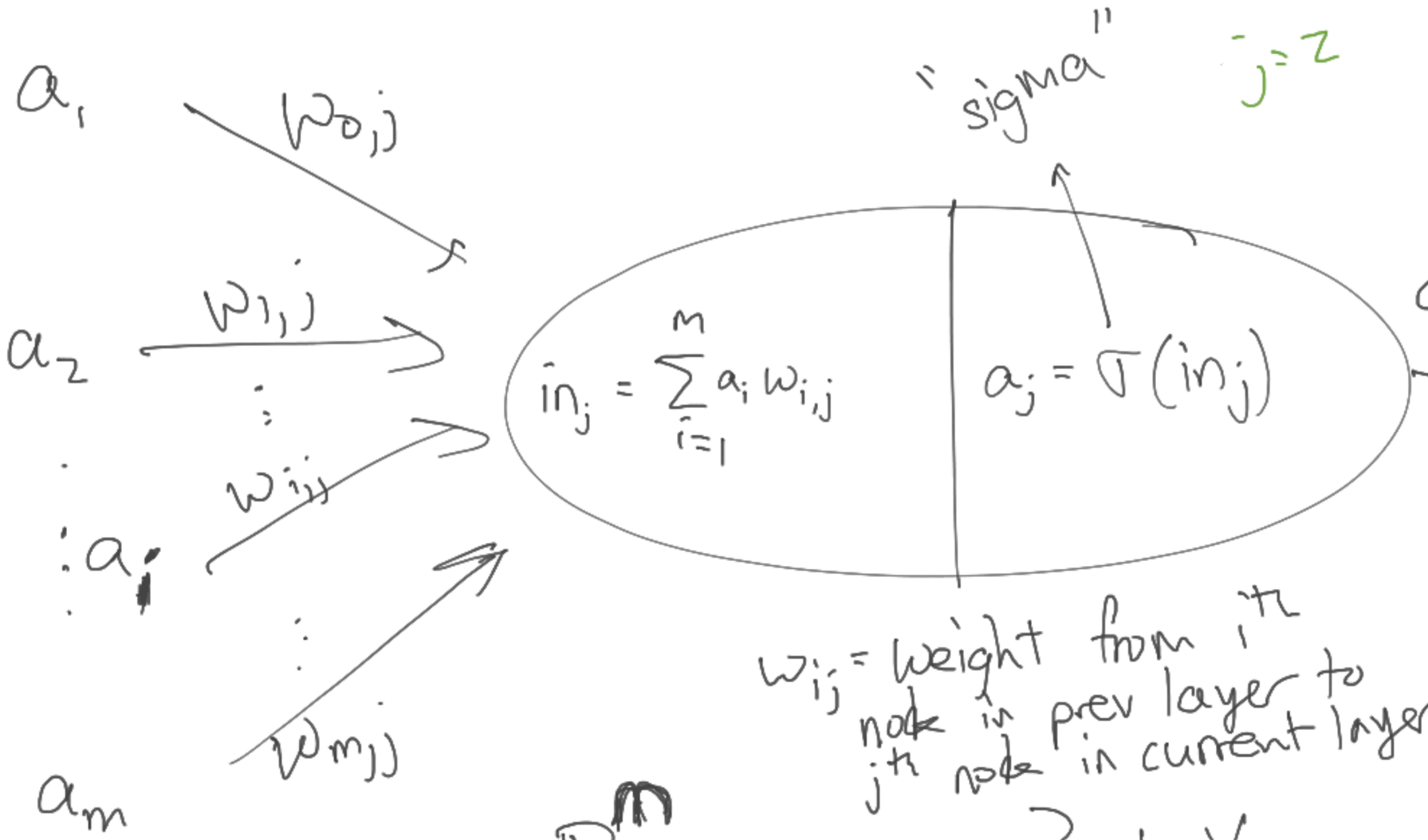
* layers
* nodes per layer (not always same)
- activation function.

New Notation

Trick: Assume "current layer" → j

"previous layer" → i

"next layer" → k



i j k

prev → current → next.

$w_{i,j}$ = weight from i^{th} node in prev layer to j^{th} node in current layer.

a_i : output of i^{th} node in prev layer.

a_j : output of j^{th} node in current layer.

$x \in \mathbb{R}^m$ ↳ input to network. } not X_i any more.

Forward Pass

// load input layer

for each input x_j in X → input to network.

$a_j \leftarrow x_j$ // a_i from input layer.

// Run hidden layers.

for each hidden layer

for each node j in current hidden layer do:

$$in_j = \sum_i a_i w_{ij}$$

$$a_j = \sigma(in_j)$$

// Return: output of output layer.

weights 4-D.

$w[l][j][i]$

- l → layer
- j → node
- i → i-th input.

$f(x)$

$x_{k+1} = x_k - \alpha \nabla f(x_k)$

$x \leftarrow x - \alpha \nabla f(x)$

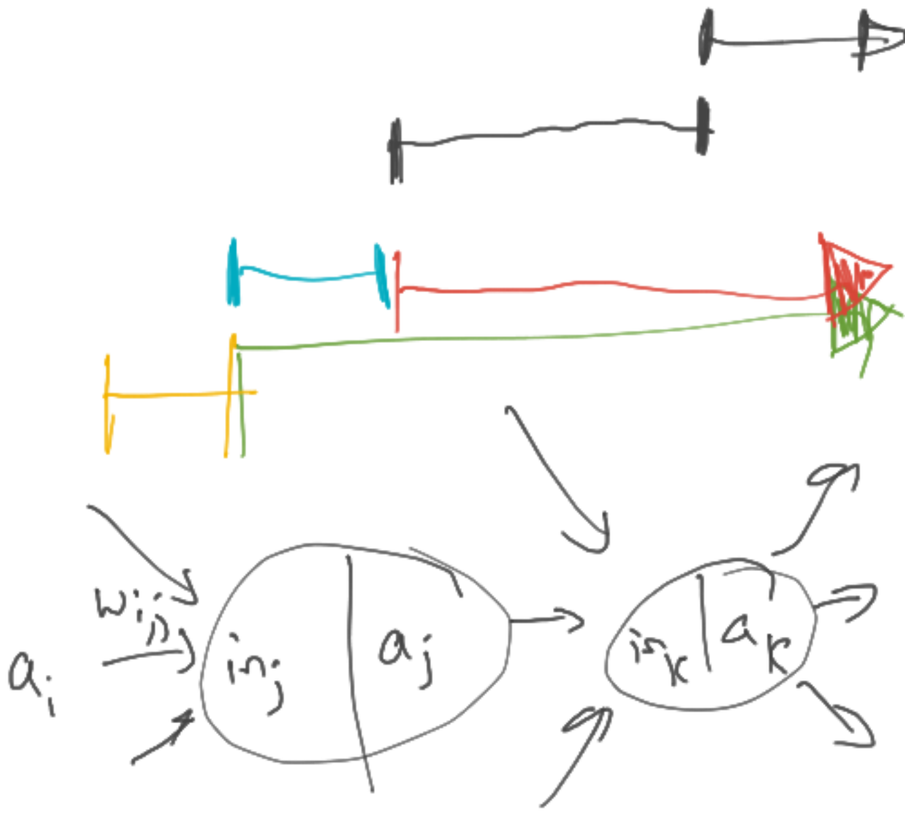
lecture 9
-Backpropagation

$$\frac{\partial f_w(x)}{\partial w_{ij}} = \frac{\partial f_w(x)}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}}$$

$$\Delta_j = \frac{\partial f_w(x)}{\partial in_j} \quad \text{(pink)}$$

$$= \frac{\partial f_w(x)}{\partial a_j} \frac{\partial a_j}{\partial in_j} \frac{\partial in_j}{\partial w_{ij}}$$

$$\Delta_j$$



$$\frac{\partial in_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k a_k w_{kj}$$

$$= a_i$$

$$\frac{\partial a_j}{\partial in_j} = \frac{\partial\, \sigma(in_j)}{\partial in_j}$$

$$= \sigma(in_j)(1 - \sigma(in_j))$$

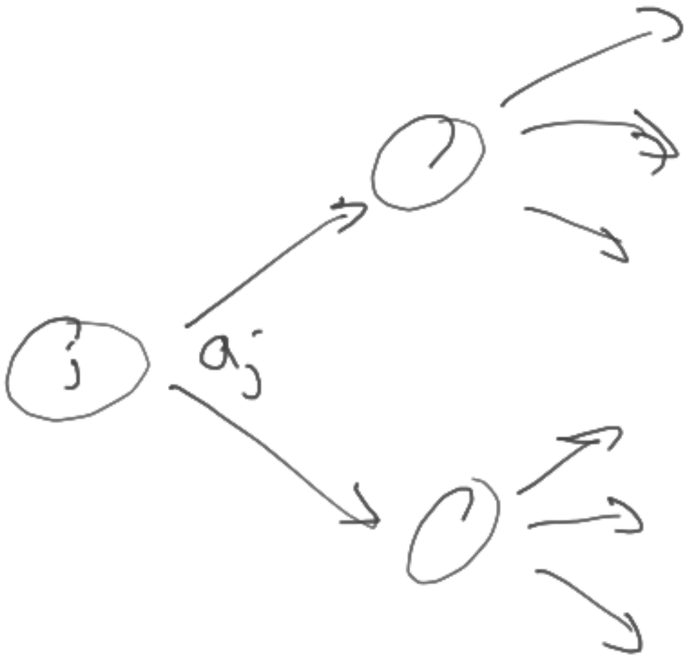
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

$$\frac{\partial f_w(x)}{\partial a_j} = \frac{\partial f_w(x)}{\partial in_k} \frac{\partial in_k}{\partial a_j}$$

$\frac{\partial in_k}{\partial a_j} = w_{jk}$

$\frac{\partial f_w(x)}{\partial in_k} = \Delta_k$

$$\frac{\partial f_w(x)}{\partial a_j} = \sum_k \frac{\partial f_w(x)}{\partial in_k} \frac{\partial in_k}{\partial a_j}$$



$$\frac{\partial f_w(x)}{\partial w_{i,j}} = \underbrace{\sum_k \Delta_k w_{j,k} \; \sigma(in_j)(1-\sigma(in_j))}_{\Delta_j} a_i$$

$a_j = \sigma(in_j)$

Δ_j for output layer?

$$\Delta_j = \frac{\partial f_w(x)}{\partial in_j} = \frac{\partial f_w(x)}{\partial a_j} \frac{\partial a_j}{\partial in_j}$$

$$= 1 \; \sigma(in_j)(1-\sigma(in_j))$$

$$w_{ij} \leftarrow w_{ij} - \alpha 2 \sum_{\substack{(x,y) \\ \text{in} \\ \text{data set}}} (y - f_w(x)) \frac{\partial f_w(x)}{\partial w_{ij}}$$

prev slide

Incremental gradient descent. (convergence similar to grad descent)

↘ For each point (x,y) in data set:

For each weight w_{ij}

$$w_{ij} \leftarrow w_{ij} - \alpha 2 (y - f_w(x)) \frac{\partial f_w(x)}{\partial w_{ij}}$$

"estimate of gradient from one point"

Backpropagation

Input: Data set consisting of many input-label pairs, (x, y)

Output: Weights w that approximately minimize the least squares loss, for a fully connected feed forward ANN.

Hyperparams: # of hidden layers, number of nodes per layer, initial weight distribution p, step size α, stopping criterion.

```
// initialize weights w
for each weight w_ij in w
    w_ij ~ p
// Run gradient descent
While stopping criterion not satisfied
    for each point (x,y) in data set
        ŷ = Forwards Pass (x, w)
        // compute Δ_j for output layer.
        for each node j in output layer:
            Δ_j = σ(in_j) (1 − σ(in_j))
```

(1) (2)

// compute Δ_j values for hidden layers.

For each hidden layer FROM OUTPUT TO INPUT

  For each node j in hidden layer

    $\Delta_j = \sum_k \Delta_k w_{jk} \sigma(in_j)(1-\sigma(in_j))$

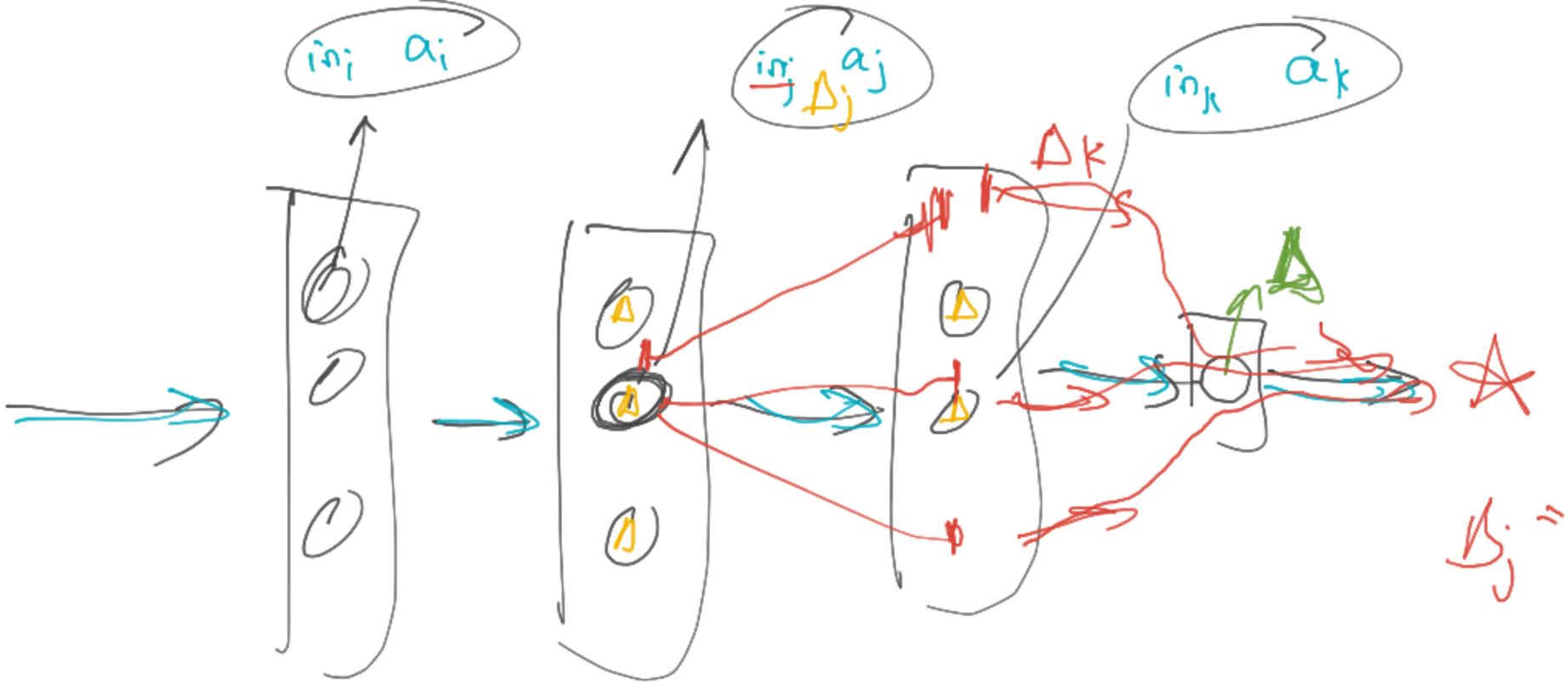
3

// perform incremental gradient step.

For each weight w_{ij} in w (all weights in all layers)

  $w_{ij} \leftarrow w_{ij} - \alpha \Sigma(y - \hat{y}) \frac{\partial f_w(x)}{\partial w_{ij}} \Delta_j a_i$

return w



$\Delta_j = \frac{\partial f_w(x)}{\partial in_j}$

(x, y)

1. Forward pass
2. Δ_j output special case.
3. Backwards pass compute Δ