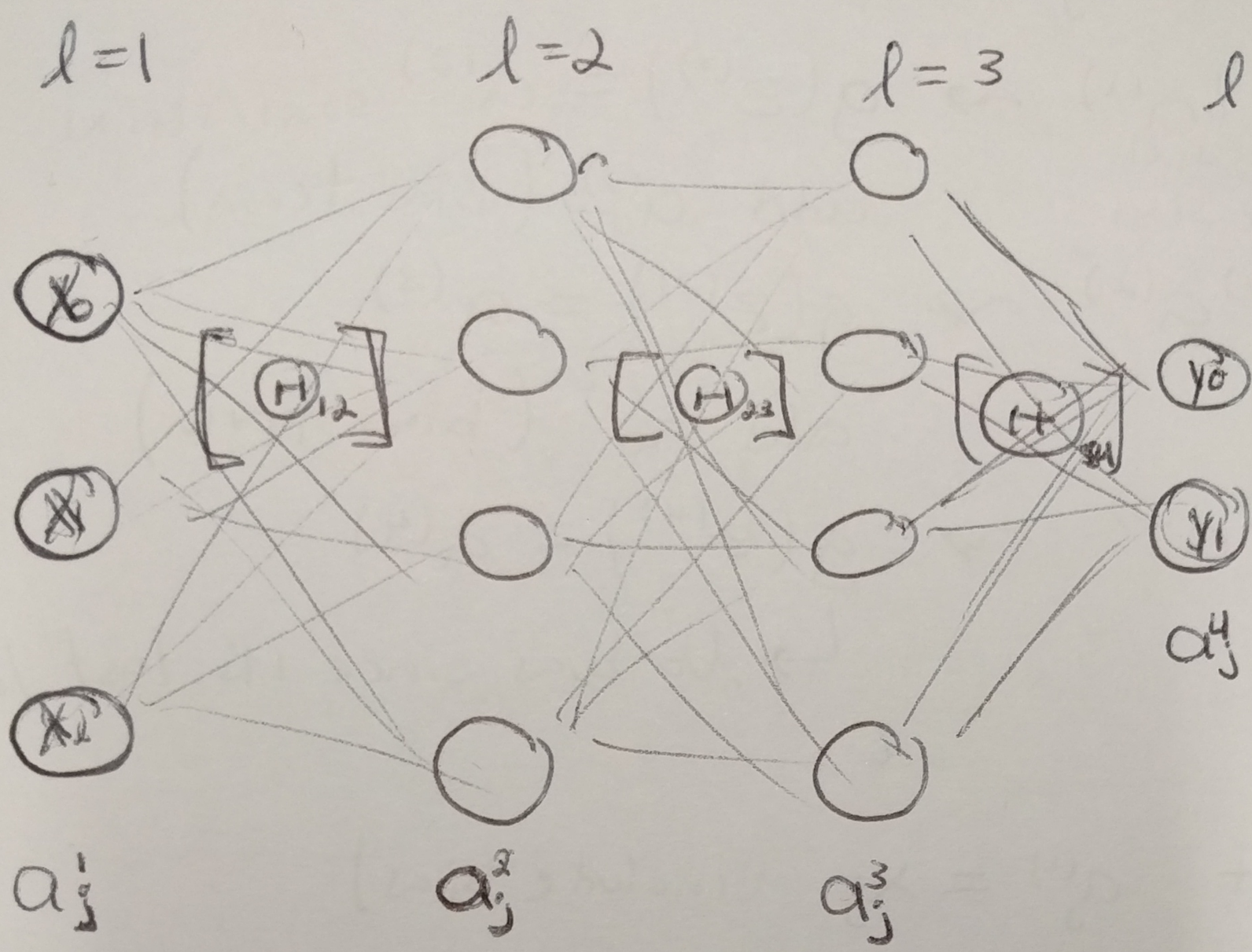
Basic FF-NN Implementation



$x^{(i)}_j$ = input from the i-th training example, describing the j-th feature.

$\Theta^{(l)}_{AB}$ = Weight matrix, mapping inputs from the l layer to the $l+1$ layer. Dims: $S_{l+1} \times S_l$

Note: $S_{l+1} \times S_l + 1$, if bias is added.

$a^{(l)}_j := g(\Theta^{(l-1)} \cdot a^{(l-1)}_j) = g(z^{(l)})$ ~ activation function value

$L :=$ number of layers

$m :=$ number of training examples.

* Feed-forward pass

- Set $a_j^{(1)} = x_j$

→ $z^{(2)} = \Theta^{(1)} a^{(1)} \rightsquigarrow g(z^{(2)}) = a^{(2)}$ $s_2 \times 1 / s_{l+1} \times 1$

$s_2 \times 1$ $s_2 \times s_1$ $s_1 \times 1$

$s_{l+1} \times 1$ $s_{l+1} \times s_l$ $s_l \times 1$

add $a_0^{(2)}$ (bias term)

→ $z^{(3)} = \Theta^{(2)} a^{(2)} \rightsquigarrow g(z^{(3)}) = a^{(3)}$

add $a_0^{(3)}$ (bias term)

→ $z^{(4)} = \Theta^{(3)} a^{(3)} \rightsquigarrow g(z^{(4)}) = a^{(4)}$

↳ No bias since it's last layer.

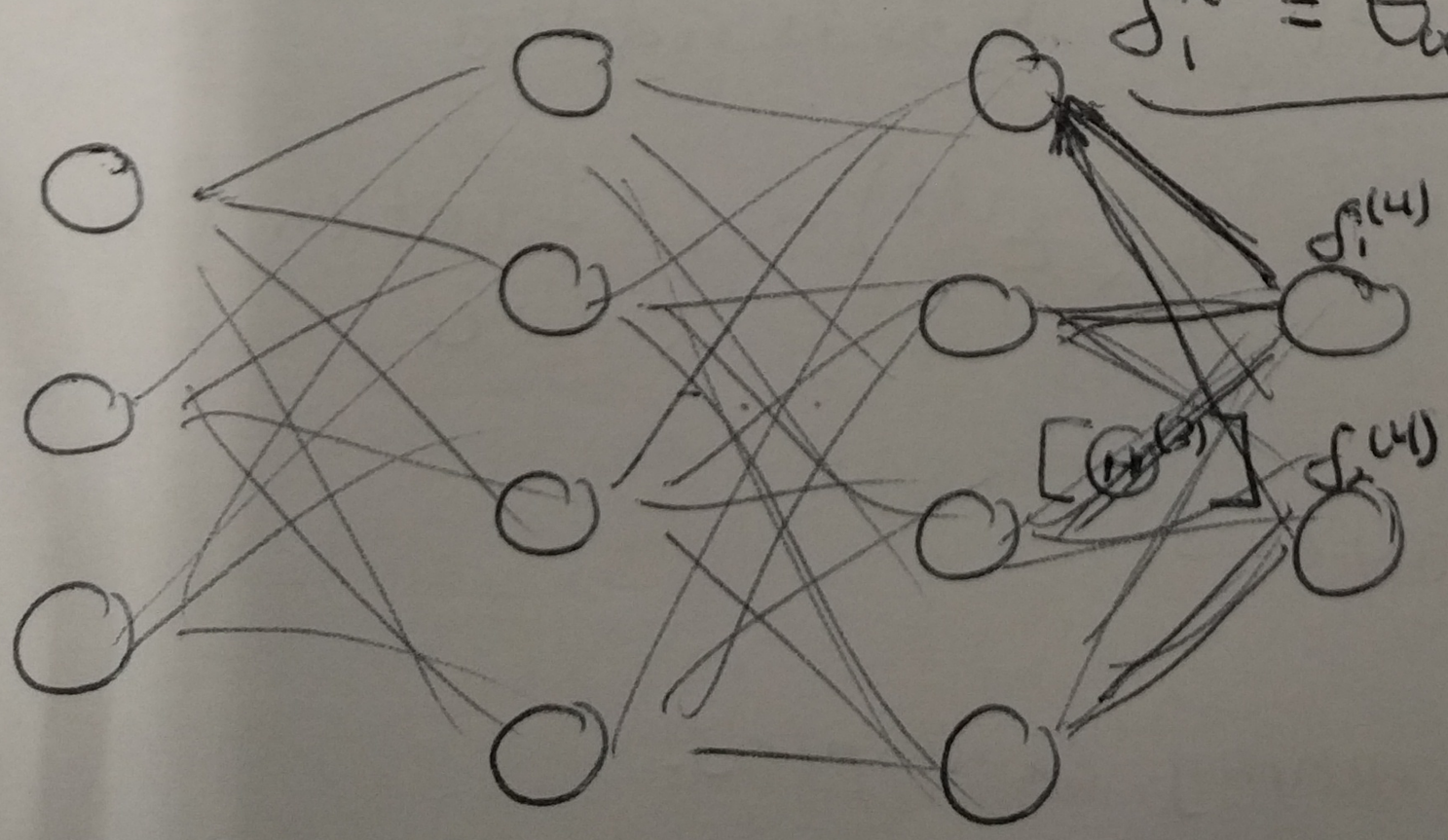
↳

- Set $a_j^{(1)} = x_j$ (include bias)
- For $l = 1$ to $l = L-1$:
  - $z^{(l+1)} = \Theta^{(l)} a^{(l)} \rightsquigarrow g(z^{(l+1)}) = a^{(l+1)}$
  - add $a_0^{(l+1)}$ (bias) ↳ if $l+1 != L$

---

* Back Propagation

$\delta_1^{(3)} = \Theta_{10} \delta_1^{(4)} + \Theta_{10} \delta_2^{(4)}$



Weighted sum of the error.

• Roughly we can express the error and derivative as follows: ③

$$E = \frac{1}{2m} \sum^{m} \sum^{K\text{-classes}} \left(g(\Theta x) - y \right)^2$$

$$\frac{\partial E}{\partial \Theta_j} \approx -\frac{1}{m} \left(y - g(\Theta x) \right) g'(\Theta x) x_j$$

$$\frac{\partial E}{\partial \Theta_j} \simeq (\text{error\_term})\, g'(\,)(\text{input})$$

↳ activation fcn. derivative.

↓

• Computation of propagated error.

$$\delta^{(4)} = a_j^{(4)} - y_j \rightsquigarrow \delta^{(L)} = \left(a_j^{(L)} - y_j \right) g'(a_j^{(L)})$$

↳ assumes there is no act. fcn. in last layer.

$$\delta^{(3)} = \left(\Theta^{(3)} \right)^T \delta^{(4)} .* g'(z^{(3)})$$

↳

$$\underset{s_{l-1}\times 1}{\delta^{(l-1)}} = \underbrace{\left(\Theta^{(l-1)} \right)^T}_{s_{l-1}\times s_l} \underbrace{\delta^{(l)}}_{s_l \times 1} .* \underset{s_{l-1}\times 1}{g'(z^{(l-1)})}$$

$$s_{l-1}\times 1 \quad .* \quad s_{l-1}\times 1$$

Mathematically it can be proved: $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$

$$\underset{s_{l+1}\times 1}{\delta^{(l+1)}} \cdot \underset{1\times s_l}{(a^{(l)})^T} = \underset{s_{l+1}\times s_l}{\Delta^{(l)}} \rightsquigarrow \text{Update increment for weight matrix.}$$

* BP - Algorithm

- Training set: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$
- Set $\Delta_{ij}^{(l)} = 0 \quad \forall_{i,j,l} \rightarrow$ Accumulators for partial derivatives

- For $i = 1$ to m:

→ Set $a^{(1)} = x^{(i)}$ [Input layer]

→ Forward Propagation → compute all $a^{(l)}$

→ $\delta^{(L)} = (a^{(L)} - y^{(i)})\, g'(z^{(L)})$ [ERROR OUTPUT LAYER]

→ Compute: $\delta^{l} \; \forall \{1 < l < L\}$

↳ $\delta^{(l)} = \underbrace{(\Theta^{(l)})^T \delta^{(l+1)} .* g'(z^{(l)})}$

Weight propagation * derivative.

→ Accummulate:

↳ $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$:= \Delta^{(l)} + \delta_i^{(l+1)} (a_j^{(l)})^T$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ (Bias term)

} Final derivative.

- Update weights:

$\Theta_{ij}^{(l)} := \Theta_{ij}^{(l)} + \alpha D_{ij}^{(l)}$

↳ learning rate